ISSN 0104-4869

REVISTA DO INSTITUTO DE

INFORMÁTICA

PUCCAMP - VOLUME 2 - Nº 2 JUNHO/DEZEMBRO/1994

CONTEÚDO/CONTENTS

Artigos/Articles

Revista do Instituto de Informática da PUCCAMP. -

Volume 1, n. 1 (1992) - . Campinas:

Semestral

1. Informática-Periódico

CDD 001.61

CDU 681.3

Revista do Instituto de Informática da PUCCAMP

Publicação Semestral

Editora-Executiva: Profª M. Cristina L. F. M. Aranha

Capa

Máscara de circuito integrado cedida pela FCTI - Fundação Centro Tecnológico para Informática.

Correspondência:

A/C:
Instituto de Informática - PUCCAMP
C. P. 317 - Campus I - Rod. D. Pedro I - Km 136 - CEP: 13020-904 - Campinas - SP - FAX: (0192) 52.8477

A "Revista do Instituto de Informática" tem uma tiragem de 2000 exemplares. É distribuída gratuitamente às Universidades, Centros de Pesquisa, Órgãos Governamentais e Empresas que nos solicitam.

Composição e Impressão

Gráfica PUCCAMP

ISSN 0104 - 4869

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

INSTITUTO DE INFORMÁTICA

REVISTA DO INSTITUTO DE INFORMÁTICA DA PUCCAMP

Rev. Inst. Inform. PUCCAMP	Campinas	V. 2	n. 2	p. 1-64	junho/dez/1994

EDITORIAL

Após receber diversas sugestões de pessoas que direta ou indiretamente colaboram (ou colaboraram) com a Revista do Instituto de Informática, o Conselho Consultivo decidiu realizar algumas mudanças que afetam principalmente a publicação da Revista. A partir deste número a Revista continuará com periodicidade semestral, publicando nos meses de junho e dezembro e não mais em março e setembro, como até então, visando, principalmente, facilitar a organização dos volumes da Revista nas bibliotecas. As outras alterações serão comunicadas à medida que forem ocorrendo.

Este número apresenta artigos técnicos e científicos, a seção de opiniões, que discute aspectos relacionados ao "backlog" (demanda reprimida) existente em organizações que trabalham com sistemas de informação, e informativos sobre os cursos de pós-graduação do Instituto de Informática.

Os cursos de especialização oferecidos pelo Instituto de Informática foram reestruturados e o curso de Mestrado, já com diversas turmas em andamento, mostra os assuntos que estão sendo desenvolvidos pelos mestrandos e seus orientadores. As primeiras dissertações serão apresentadas ainda este ano.

Como acontece em todo o número da Revista,que sempre traz um artigo sobre Gerenciamento de Sistemas de Informação, neste número o artigo apresentado considera alguns aspectos dos sistemas de apoio à tomada de decisões em grupo, no gerenciamento das modernas organizações.

O paradigma de orientação a objeto é discutido em dois artigos. Em um deles analisa-se o particionamento de sistemas, principalmente em casos que sejam complexos ou que tenham um conjunto diversificado de comportamentos. No outro artigo é feita uma associação com inteligência artificial e apresenta-se uma metodologia para auxiliar na especificação, orientada a objeto, de sistemas de manufatura.

A importância de interfaces adequadas para ambientes de programação é discutida em um dos artigos, que também descreve carcterísticas de algumas interfaces existentes que apoiam a atividade de programação.

Há um artigo que apresenta uma proposta de modelo para auxiliar na solução de problemas que ocorrem em sistemas distribuídos abertos (trabalhando com um objeto computacional denominado TRADER), discute a união de traders em federações e descreve, com detalhes, uma implementação de trader que está sendo feita na UNICAMP.

Há, ainda, um artigo que descreve o desenvolvimento de um sistema para geração automática de programas para a inserção de componentes em placas de circuito impresso, utilizando arquivo CAD.

O trabalho por mim realizado, como Editora Executiva da Revista, foi árduo, parecendo,algumas vezes, ser quase impossível transpor todas as dificuldades, mas foi muito gratificante. É preciso, porém, que outros docentes do Instituto exerçam essa função, pois a renovação é necessária, além de ser muito saudável. Espero que o próximo número apresente essa renovação e, como não poderia deixar de ser, devo agradecer a muitas pessoas que, de alguma maneira, contribuíram para o êxito que a Revista tem tido até agora.

Agradeço a todos aqueles que colaboraram com artigos, opiniões, informativos e entrevistas, além, é claro, das sugestões, apresentados neste e nos números anteriores da Revista. Agradeço, também, à Direção e Vice-Direção do Instituto de Informática, pois não mediram esforços para garantir as publicações dos números da Revista.Não posso deixar de agradecer aos membros do Conselho Editorial, pois exerceram papel preponderante na determinação do conteúdo a ser publicado, através da análise de artigos a eles submetidos. Agradeço,também, aos membros do Conselho Consultivo que me deram todo o apoio necessário, principalmente nas horas de maior dificuldade que naturalmente existem nesse tipo de trabalho. Agradeço não só à Administração Superior da Universidade - Reitor Prof. Gilberto Luiz M. Selber, Prof. Alberto Martins, Vice-Reitor para Assuntos Administrativos e Pe. José Benedito A. David, Vice-Reitor para Assuntos Acadêmicos, bem como à IBM Brasil, que acreditaram no trabalho do Instituto de Informática. Finalmente, não poderia deixar de dizer que as pessoas de apoio à infra-estrutura da Revista (pessoas responsáveis pela distribuição, secretárias, digitadores, diagramadores, revisores e montadores da Revista) foram de fundamental importância para garantir o sucesso da Revista.

A todas as pessoas citadas acima e àquelas que por alguma falha deixei de citar MUITO OBRIGADA !

Profª M. Cristina L. M. Aranha
Editora Executiva

CONTEÚDO/CONTENTS

Artigos/Articles

OS SISTEMAS DE APOIO À TOMADA DE DECISÕES EM GRUPO (GDSSs) NO GERENCIAMENTO DAS MODERNAS ORGANIZAÇÕES

GROUP DECISION SUPPORT SYSTEMS (GDSSs) IN THE MANAGEMENT OF MODERN ORGANIZATIONS

Prof. Dr. Juan Manuel Adán COELLO*

ABSTRACT

Current trends, including both managerial and technological issues, are leading the work of groups to be come a key aspect within modern organizations, particularly when complex decisions concerning unstructured problems have to be taken. It is well known that the problem solving abilities of a group working in synergy can be significantly higher then that of its individual members. However, it is also no known that groups can suffer from several disfunctions that can turn their work very little productive. Group Decision Support Systems (GDSSs) are made up of hardware, software, people and procedures that support the work of groups in decision related meetings, aiming at the reduction of the time and cost of taking effective decisions. This paper discusses the main concepts of GDSS and some of the trends in the area, in order to contributed to the identification of research opportunities and to the understanding of the rule that GDSSs could play in the management of modern organizations.

KEY WORDS: Group Decision Support Systems (GDSS); Computer Supported Cooperative Work (CSCW); Groupware; Management Support Systems (MSS);

RESUMO

Tendências atuais, envolvendo tanto aspectos gerenciais como tecnológicos, estão tornando cada vez mais importante o trabalho em grupo dentro das organizações, particularmente quando decisões complexas, envolvendo problemas não estruturados, devem ser tomadas. É sabido que quando o trabalho em grupo produz sinergia, a sua capacidade de resolução de problemas em muito supera as capacidades dos indivíduos que o formam. No entanto, é também sabido que os grupos podem sofrer de diversas disfunções que podem tornar o seu trabalho muito pouco produtivo. GDSSs são uma combinação de *hardware, software,* pessoas e procedimentos, que suportam o trabalho de grupos em encontros voltados à tomada de decisões, com o objetivo de reduzir o tempo e o custo necessários à tomada de decisões eficazes. Este artigo discute os principais conceitos de GDSSs e aponta as suas perspectivas de evolução. Procura-se, com isso, contribuir para a identificação de oportunidades de atuação na área, bem como para o entendimento do papel que os GDSSs podem assumir no gerenciamento das modernas organizações.

PALAVRAS-CHAVE: Sistemas de Apoio à Tomadas de Decisões em Grupo (GDSSs); Trabalho Cooperativo Suportado por Computador (CSCW); Groupware; Sistemas de Suporte ao Gerenciamento (MSSs).

1. INTRODUÇÃO

A maior parte das vezes que uma decisão importante deve ser tomada numa organização, é constituído um grupo a fim de tomar essa decisão, ou aconselhar e assessorar ao responsável por fazê-lo. Por outro lado, tendências atuais, como a introdução de programas de aumento de qualidade, a implantação do conceito de Engenharia Simultânea (ES), e, segundo muitos autores, a própria introdução de sistemas computadorizados (George and King, 1991), estão levando as empresas a assumirem uma estrutura de rede, em lugar da tradicional estrutura hierárquica. Estas organizações, mais capazes de lidar com mudanças, sejam elas de atividades, de tecnologias ou de ambientes, aptas, portanto, a aprender ("*learning organizations*"), delegam responsabilidade

(*) Pesquisador da Fundação Centro Tecnológico para Informática - Professor do Instituto de Informática da PUCCAMP.

por tomar decisões a um número crescente de indivíduos, freqüentemente trabalhando em grupo.

Um Sistema de Apoio à Tomada de Decisões em Grupo (*Group Decision Support System - GDSS*) é uma combinação de tecnologias emergentes, incluindo redes de computadores, multimídia e inteligência artificial, e de técnicas desenvolvidas nas áreas de ciência do gerenciamento, psicologia, e ciências sociais, que visa suportar o processo de tomada de decisões envolvendo grupos de indivíduos. Apesar de tratar-se de uma área de pesquisa relativamente recente, ela tem despertado o interesse de um considerável número de grupos de pesquisa, havendo inclusive, alguns produtos já disponíveis no mercado, tendência que deve intensificar-se em função dos recentes avanços das tecnologias de computação e informática que lhe dão suporte.

Este artigo faz uma introdução à área de GDSSs, procurando oferecer os elementos básicos que permitam avaliar algumas oportunidades de atuação na área, bem como contribuir para o entendimento do papel que os GDSSs podem assumir na gerência das organizações modernas. Para tanto, o artigo está organizado da seguinte maneira: na seção 2, serão apresentados os principais benefícios e problemas usualmente associados ao trabalho de grupos; em seguida, na seção 3, é feita a caracterização de um GDSS, em termos da sua configuração e funcionalidade; na seção 4, apresenta-se, a título ilustrativo, um exemplo clássico de GDSS; finalmente, na seção 5, são feitas algumas considerações finais e sinalizadas algumas linhas de pesquisa que requerem ainda considerável atenção, de modo a possibilitar a construção dos GDSSs do futuro.

2. VANTAGENS E DESVANTAGENS ASSOCIADAS À TOMADAS DE DECISÕES EM GRUPO

Associa-se ao trabalho em grupo em geral, e à tomada de decisões em particular, diversos aspectos positivos, entre eles, Nunamaker et al. (1991) destacam os seguintes:

- grupos têm mais facilidade para entender problemas que indivíduos isoladamente;
- o grupo como um todo é detentor de mais informação e conhecimento que qualquer um de seus membros individualmente e, em conseqüência, tem mais elementos para resolver um problema;
- uma informação apresentada ao grupo tende a receber novas interpretações e usos, em função das diferentes informações e habilidades já possuídas por cada um dos seus membros (produção de sinergia);
- trabalhar em grupo pode estimular e encorajar os indivíduos a aumentar o seu desempenho;
- a participação dos indivíduos numa decisão contribui para reduzir a resistência à sua implementação.

Por outro lado, o trabalho em grupo está também associado a diversas disfunções e entre elas encontram-se as seguintes:

- a dominação de alguns membros do grupo que exercem uma influência indevida ou monopolizam o tempo do grupo de maneira não produtiva, devido a características de personalidade ou status;
- problemas de coordenação, decorrentes da falta de uma estratégia apropriada para integrar as contribuições dos membros do grupo podem levar a decisões prematuras, fruto de discussões redundantes e incompletas;
- discussão de assuntos não relacionados à tarefa em questão (sociabilização) reduz o desempenho do grupo, embora alguma sociabilização seja normalmente necessária para o efetivo funcionamento de um grupo;
- acesso limitado e uso incompleto da informação necessária ao desempenho da tarefa em questão;
- tendência de alguns membros do grupo em deixar que os outros façam a maior parte do trabalho;
- o receio de avaliações negativas pode conduzir alguns membros a não expressar suas próprias idéias ou não opinar a respeito de idéias alheias;
- membros podem relutar em criticar os comentários dos outros, por delicadeza ou medo de represálias;
- fragmentação do tempo de reunião entre os participantes;
- um indivíduo, ao não poder expressar os seus comentários assim que lhe ocorrem (por já haver alguém fazendo uso da palavra), pode, posteriormente, esquecê-los ou suprimí-los (por lhe parecerem menos originais, relevantes ou importantes);
- redução do número de comentários em função de os membros do grupo deverem se concentrar em lembrar dos seus comentários até poder fazê-los, em lugar de pensar em novas contribuições;
- novos comentários não são gerados porque os membros devem ouvir constantemente aos outros e não podem parar para pensar.

A fim de reduzir as perdas inerentes ao trabalho em grupo acima enumeradas, pesquisas nas áreas de psicologia e ciência do gerenciamento, entre outras, produziram algumas metodologias e técnicas, como, por exemplo, a NGT (*Nominal Group Technique*) e o método Delphi (Turoff, M. and S. R. Hiltz, 1982). Embora a utilização dessas técnicas se mostre, de modo geral, benéfica, o seu emprego costuma ser associado a custos elevados e decisões demoradas.

3. CONCEITUAÇÃO DE GDSSs

A importância da tomada de decisões em grupo, as diversas disfunções associadas ao trabalho em grupo e o limitado sucesso das técnicas manuais empregadas para reduzir essas disfunções, juntamente com o barateamento das tecnologias de comunicação e computação, têm levado a um crescente interesse por sistemas computacionais que suportem o trabalho de grupos.

DeSanctis e Gallupe (1987) definem um GDSS como um sistema que combina tecnologias de comunicação, computação e tomada de decisões para suportar a formulação e solução de problemas em encontros de grupos de indivíduos (tomadores de decisões).

GDSSs diferem-se dos Sistemas de Apoio à Tomada de Decisões (DSS) convencionais por centrarem-se mais no processo de tomada de decisão pelo grupo, que na solução de um problema específico. Por outro lado, GDSSs são diferentes da categoria mais abrangente de Sistemas Computacionais de Suporte ao Trabalho Cooperativo (CSCW), por se concentrarem fundamentalmente em questões associadas à tomada de decisões em horizontes relativamente curtos (Connolly et al., 1990).

3.1. CARACTERIZAÇÃO DE GRUPOS

Para caracterizar um grupo, sob a ótica de um GDSS, devem ser consideradas quatro variáveis principais: a dispersão geográfica do grupo; a dispersão temporal das interações entre os membros do grupo; o tamanho do grupo e o tipo de tarefa a ser realizada pelo grupo.

DISPERSÃO GEOGRÁFICA DO GRUPO

Em termos de sua dispersão geográfica, um grupo pode ser "co-localizado", quando os seus membros se reúnem num mesmo local, ou remoto, quando os seus membros se encontram em distintos locais. Para alguns autores, como Roden e Blair (1992), a distinção entre grupos co-localizados e remotos deve levar em consideração não somente a distância física entre os seus membros, como também, e especialmente, a sua distância lógica, função dos mecanismos de interação existentes. Sob esta ótica, podem ser definidas quatro categorias de grupos:

- *CO-LOCALIZADOS*: todos os membros do grupo se reúnem num mesmo local. Sistemas desta categoria são normalmente implementados sob a forma de uma sala de reuniões com estações de trabalho interligadas em rede e uma grande tela pública.
- *VIRTUALMENTE CO-LOCALIZADOS*: funcionalmente similares aos anteriores, porém os membros do grupo não estão fisicamente num mesmo local. Esse tipo de sistema pode ser viabilizado pela utilização de tecnologias de multimídia, apoiadas em canais de comunicação que suportem a transmissão em tempo-real de dados, voz e vídeo.
- *"LOCALLY REMOTE"*: assumem que os membros do grupo estão localizados num mesmo edifício, o que reduz os custos para sua interação.
- REMOTOS: pressupõem facilidades mínimas de interação entre os membros do grupo.

DISPERSÃO TEMPORAL

GDSSs podem ainda ser síncronos, assíncronos ou uma combinação de ambos, em função da dispersão temporal das interações entre os membros do grupo. Num sistema síncrono, as interações ocorrem com a presença simultânea de todos os membros do grupo. Em sistemas assíncronos, o trabalho do grupo desenvolve-se ao longo de períodos mais longos, não sendo necessária a presença de todos os membros do grupo ao mesmo tempo. Finalmente, em sistemas mistos, os trabalhos do grupo desenvolvem-se ora com a interação simultânea de todo o grupo, ora com os elementos do grupo trabalhando independentemente.

TAMANHO DO GRUPO

Em função do número de integrantes, grupos são genericamente classificados como pequenos, médios e grandes. A quantificação do número de indivíduos em cada uma dessas classes de grupos é, no entanto, muito dependente do tipo de tarefa a ser conduzida.

Estudos feitos em diversas disciplinas das ciências humanas, assim como resultados de experimentos envolvendo a utilização de sistema de comunicação eletrônica e suporte computacional à decisão em grupo, sugerem que a natureza da troca de informação e os resultados do processo de tomada de decisão são fortemente afetados pelo tamanho do grupo (Hiltz and Turof, 1978). Por outro lado, o tamanho ideal de um grupo depende de inúmeros fatores (o grupo em si, o contexto em que o trabalho se desenvolve, etc) e, em alguns casos, pode ser razoavelmente grande, ao contrário do que normalmente se acredita (Nunamaker et al., 1991).

TAREFAS DO GRUPO

Além dos fatores situacionais mencionados, a tarefa a ser desempenhada por um grupo, ou seja, o objetivo a ser atingido pelo grupo no curso de uma reunião, é um dos aspectos importantes a considerar no projeto de um GDSS. Segundo McGrath (1984), citado por DeSanctis e Gallupe (1987), os principais objetivos de um grupo, numa reunião voltada à tomada de decisões, incluem: a **geração** de idéias e ações, associadas tanto a tarefas de planejamento, que requerem a geração de planos orientados a ações a tomar, como a tarefas criativas, que requerem novas idéias; a **escolha** de alternativas, baseada em um processo de manifestação de preferências, quando não houver um critério objetivo para avaliar a correção da escolha; e a **negociação** de soluções, que pode implicar na superação de conflitos decorrentes de pontos de vista distintos (conflito cognitivo) ou de interesses divergentes.

3.2. NÍVEIS DE SUPORTE À DECISÃO EM GRUPO

DeSanctis e Gallupe (1987) agrupam as facilidades que podem ser oferecidas por um GDSS em três níveis. As **facilidades de nível 1** são normalmente encontradas em "sistemas de reunião eletrônica" ("Electronic Meeting Systems-EMS") e visam fundamentalmente facilitar a troca de informação entre os elementos do grupo. Facilidades de nível 1 incluem:

- Troca de mensagens entre os elementos do grupo.
- Interconexão das estações de trabalho dos membros do grupo entre si e aos sistemas de gerenciamento de dados da empresa, de modo a permitir o acesso aos dados pessoais ou corporativos durante uma reunião.
- Uma grande tela visível a todos os participantes, ou uma janela "pública" no terminal de cada participante, para que possam ser apresentadas idéias, votos, dados, gráficos ou tabelas a todos os participantes simultaneamente.
- Entrada anônima de idéias e votos, de modo a fazer frente à relutância de alguns membros em se expressar, devido a sua timidez, baixo status ou idéias controvertidas.
- Solicitação ativa de idéias e votos de cada membro do grupo para encorajar participação e induzir criatividade.
- Sumário e apresentação de idéias e opiniões, incluindo sumários estatísticos e contagem de votos (na tela/janela pública), para contribuir com a organização e análise das idéias e votos já manifestados.
- Esquemas de pontuação e priorização, para facilitar a quantificação das preferências dos elementos do grupo.
- Um esboço de agenda, a ser completada pelo grupo, para ajudar na organização de estratégias e planos para a condução de reuniões.
- Apresentação automática de itens da agenda em instantes apropriados da reunião para contribuir com o cumprimento da programação elaborada.

As **facilidades de nível 2** oferecem técnicas de modelagem e análise para suportar a tomada de decisões. Um GDSS de nível 2 pode oferecer ferramentas para suporte ao planejamento e controle, como CPM ou PERT, ou outras ferramentas normalmente encontradas em DSS para trabalho individual. Num GDSS, no entanto, as ferramentas devem poder ser utilizadas pelo grupo cooperativamente e os resultados visualizados simultaneamente por todos. Além dessas ferramentas, GDSSs de nível 2 podem automatizar técnicas manuais de estruturação do trabalho em grupo, como o método Delphi e a NGT ("Nominal Group Tecnhique") (Turoff and Hitz, 1982).

GDSSs com **facilidade de nível 3** encontram-se ainda em fase de pesquisa e se caracterizam pela automação dos padrões de comunicação do grupo, utilizando técnicas de inteligência artificial. Em sistemas com facilidade de nível 3, podem, por exemplo, ser definidas regras que determinem a seqüência de intervenção dos membros do grupo, os tipos de respostas apropriadas e as regras de votação.

4. EXEMPLO DE GDSS

O GroupSystem, desenvolvido na Escola de Pós-Graduação em Gerenciamento, da Universidade do Arizona (EUA) (Nunamaker et al,1991), é um exemplo típico de GDSS.

Esse sistema tem sido usado para suportar grupos grandes trabalhando face a face no mesmo lugar, isto é, grupos co-localizados síncronos. O GroupSystem se caracteriza por três elementos básicos: uma sala de reuniões, um facilitador e um conjunto de ferramentas de *software*.

A sala de reuniões consiste de um conjunto de microcomputadores, dotados de discos rígidos, monitores coloridos e interligação em rede. A cada participante da reunião é alocado um microcomputador próprio, a ao facilitador, ou líder, da reunião é alocado um ou dois microcomputadores para servirem de console. Adicionalmente, a sala de reuniões dispõe de um grande painel público (*eletronic blackboard*) e, opcionalmente, de outros recursos audiovisuais.

O papel de facilitador, ou líder, da reunião pode ser exercido pelo líder efetivo do grupo, por um membro qualquer do grupo ou, mais freqüentemente, por um elemento neutro, não pertencente ao grupo. O facilitador é responsável por oferecer o suporte técnico necessário à utilização do sistema. Antes da reunião o facilitador, juntamente com o grupo ou seu líder, identificam os objetivos da reunião, desenvolvem uma agenda para atingi-los e selecionam as ferramentas que devem ser usadas em cada fase da reunião. Posteriormente, o facilitador preside a reunião, procurando manter a agenda e avaliar a necessidade de alterá-la.

A qualidade do facilitador é decisiva para que os objetivos da reunião sejam atendidos, particularmente quando estiverem envolvidos grupos grandes. A automação das funções do facilitador é um interessante tema de pesquisa, associado à criação de facilidades de nível 3, para o qual a utilização de técnicas de Inteligência Artificial mostra-se promissora.

O GroupSystems oferece um conjunto de ferramentas para suportar diversas atividades do trabalho em grupo. Essas ferramentas podem ser combinadas de diversas maneiras, de modo a suportar vários estilos de reuniões. As principais classes de atividades suportadas pela ferramenta são: gerenciamento de sessões, interação do grupo e manutenção da memória organizacional.

O **gerenciamento de sessões** é feito usando uma ferramenta denominada *Session Manager* (SM). O SM suporta não só o planejamento prévio da sessão através de um questionário eletrônico, para procurar garantir que informações de planejamento não sejam esquecidas, como uma ferramenta para o auxílio na elaboração de agenda. Durante as sessões, SM é usado para iniciar todas as ferramentas necessárias, assim como para registrar informações relativas às tarefas delegadas a membros da equipe. Após as sessões, o SM permite gerar documentos que identificam os resultados da reunião e as tarefas delegadas aos seus participantes. O SM permite também organizar os resultados das sessões a fim de manter a memória organizacional.

As ferramentas de **suporte à interação do grupo** permitem conduzir três estilos de reuniões , a saber : dirigidas (*chauffered*), suportadas e interativas. Nas reuniões dirigidas, apenas uma pessoa usa o sistema, podendo ser tanto um membro do grupo como o facilitador. As discussões são feitas verbalmente e o painel eletrônico é usado como uma memória do grupo, onde as informações são registradas e estruturadas. Nas reuniões suportadas, cada membro do grupo dispõe de uma estação de trabalho, propiciando contribuições paralelas e anônimas à memória do grupo. A interação entre os elementos do grupo ocorre ora verbalmente ora eletronicamente. O painel eletrônico continua a ser usado, porém, agora, qualquer membro do grupo pode ali adicionar itens. Embora o painel eletrônico possa ser usado, a memória do grupo tende a ser muito grande para nele poder ser completamente apresentada, de modo que ela é armazenada internamente, podendo cada membro acessá-la a partir da sua estação de trabalho.

Em função do estilo de reunião sendo conduzido e da atividade a realizar, podem ser utilizadas ferramentas para geração e exploração de idéias, para a organização de idéias, para a priorização de alternativas e para o desenvolvimento e avaliação de políticas.

Finalmente, as ferramentas para manutenção da **memória organizacional** permitem capturar as adições de cada reunião à memória da organização e acessá-las em encontros subseqüentes. Essa memória é armazenada numa combinação de arquivos de texto, bases de dados e bases de conhecimento.

5. CONSIDERAÇÕES FINAIS

As empresas modernas tendem a depender, cada vez mais, das decisões tomadas por grupos de indivíduos. Experimentos realizados com GDSSs sugerem que eles podem contribuir consideravelmente para a redução do tempo necessário à tomada dessas decisões, bem como aumentar a sua qualidade e facilitar a sua implementação.

No entanto, os GDSSs ainda estão em sua infância. Apesar de já haver diversos protótipos em uso em laboratórios de pesquisa e alguns produtos disponíveis no mercado, há ainda uma série de questões que não foram satisfatoriamente resolvidas. Embora diversos estudos tenham mostrado que os GDSSs levam à produção de resultados diferentes dos obtidos sem seu uso, não foi possível explicar completamente porque isso ocorre. Não está também claro quais componentes um GDSS deve oferecer para suportar cada uma das diversas situações criadas pelo trabalho em grupo. A elucidação dessas questões, assim como a consolidação das bases conceituais para o desenvolvimento dos sistemas de suporte ao trabalho em grupo do futuro, dependem de um melhor entendimento dos padrões de cooperação que se manifestam no trabalho em grupo, o que somente poderá efetivamente ser feito por equipes multidisciplinares .

A importância de constituir equipes multidisciplinares no desenvolvimento de sistemas cooperativos pode ser claramente ilustrada pelos trabalhos de BENTLEY e colegas (1992), e SOMMERVILLE e colegas (1993). Discute-se, nesses trabalhos, o projeto de um sistema cooperativo de decisões em tempo-real para o controle do tráfego aéreo, desenvolvido por uma equipe integrada por engenheiros de *software* e sociólogos. Os estudos

etnográficos, conduzidos pelos sociólogos, revelaram padrões de cooperação de tal modo complexos e sutis que dificilmente teriam sido identificados pelos métodos estruturados, tradicionalmente empregados na análise de requisitos de sistemas. Os estudos indicaram, também, que a introdução de novas tecnologias para suportar o trabalho cooperativo deve ser cuidadosamente gerenciada, pois, do contrário, pode produzir efeitos negativos, particularmente, em termos de segurança.

O desenvolvimento dos GDSSs do futuro requer ainda um razoável volume de pesquisa para suportar o processo de negociação, aspecto central à tomada de decisão em grupo. O uso de sistemas especialistas e de outras abordagens de inteligência artifical, como CBR ("Case Based Reasoning"), e inteligência artificial distribuída, como sistemas *blackboard* e sistemas multiagentes, são promissores nessa área. Cabe destacar que alguns sistemas de suporte à negociação já foram desenvolvidos, limitando, geralmente, a lidar com situações onde há conflitos de interesses, pouco tendo sido feito para tratar os casos em que há conflitos cognitivos, isto é, diferenças de entendimento de uma situação ou problema (v. Boland et al., 1992).

O uso dos resultados das pesquisas desenvolvidas nas áreas de banco de dados e inteligência artificial serão também importantes para a manutenção e recuperação da memória organizacional. A memória organizacional permite, entre outras coisas, buscar situações ou problemas vivenciados no passado, elementos similares a uma situação ou problema presente, de tal modo que o processo de solução empregado anteriormente possa auxiliar no entendimento e solução da situação ou problema atuais (Sycara e Lews, 1991).

Cabe destacar que a maior parte dos estudos conduzidos para avaliar o GDSS,relatados na literatura, envolve sistemas que suportam grupos reunidos num mesmo local e ao mesmo tempo (co-localizados e síncronos), ou grupos trabalhando em locais distintos, em diferentes instantes (remotos e assíncronos). Esse fato pode ser explicado, em parte, pelos recursos computacionais e de comunicações existentes quando do desenvolvimento dos GDSSs em questão, mais concretamente, estações de trabalho, sem recursos de multimídia, interligadas por redes locais de média velocidade e redes de longa distância privadas e, principalmente públicas, de baixa velocidade (Bitnet, Internet, etc).

A nova geração de GDSS será, com certeza, fortemente influenciada pelas novas tecnologias de multimídia e de redes de comunicação de alto desempenho. Entre outras coisas, essas tecnologias devem levar a um maior interesse pelo desenvolvimento de sistemas de suporte a grupos virtualmente co-localizados. Por outro lado, as plataformas emergentes de suporte à construção de sistemas distribuídos abertos, tanto no contexto intra-empresa, como no contexto inter-empresa, são também um elemento fundamental a considerar na concepção do GDSS do futuro. Essas tecnologias abrem uma ampla gama de possibilidades, difícil de avaliar em toda a sua abrangência neste momento.

REFERÊNCIAS

Bentley, R., J. A. Hughes, D. Randall and D. Z. Shapiro, "Technological Support for Decision Making in a Safety Critical Environment", 1st World Congress on Safety of Transportation, Delft, The Netherlands, 1992.

Boland, R. J., A. K. Maheshwari, D. Te'eni, D. Schwartz and R. V. Tenkasi, "Sharing Perspectives in Distributed Decision Making", ACM Conference on Computer-Supported Cooperative Work, 1992.

DeSanctis G. and R. B. Gallupe. "A Foundation for the Study of Group Decision Support Systems", Management Science, vol. 33. Nº 5, May 1987.

George, J. F. and J. L. King, "Examining the Computing and Centralization Debate", Comm. of the ACM, Vol. 34, Nº 7, July 1991.

Kraemer, K. L. and J. L. King. "Computer-Based Systems for Cooperative Work and Group Decision Making", ACM Computing Surveys, Vol. 20. Nº 2, June 1988.

McGrath, J. E., "Groups: Interaction and Performance", Prentice Hall, Englewood Clifs, NJ, 1984.

Nunamaker, J. F., A. R. Demis, F. S. Valacich, D. R. Vogel and J. F. George, "Electronic Meeting Systems to Support Group Work", Communications of The ACM, vol. 34, nº 7, July 1991.

Rodden, T. and G. S. Blair. "Distributed system support for computer supported cooperative work", Computer Communications, vol. 15. nº 8, October 1992.

Sommerville, I., T. Rodden, P. Sawyer, R. Bentley and M. Twidale, "Integrating Ethnography into the Requirements Engineering Process", Proc. IEEE International Symposium on Requirements Engineering, San Diego, California, Jan. 1993.

Sycara, K. P., C. M. Lewis, "Modeling Group Decision Making and Negotiation in Concurrent Product Design". International Journal of Systems Automation: Research and Applications (SARA) 1, 217-238, 1991.

Turoff, M. and. S. R. Hiltz, "Computer Support for Groups versus Individual Decisions", IEEE Transactions on Communications, vol. 30, nº 1 (January 1982), 82-90.

PARTIÇÃO DE SISTEMAS NO PARADIGMA DE OBJETO

SYSTEM PARTITIONING IN OBJECT-ORIENTED PARADIGM

Profª Drª Beatriz M. DALTRINI*
Sergio R. SIGRIST**

ABSTRACT

Several mechanisms of system partitioning in object-oriented paradigm are available in literature, but there is not a consensus about which of them shows a more granularity context vision than a class of objects provides. This paper surveys three mechanisms often quoted: *subject*, *ensemble* and *subsystem*.

KEY WORDS: Systems Partitioning, Object Grouping, Object-Oriented Analysis.

RESUMO

Diversos mecanismos de partição de sistemas no paradigma de orientação a objetos são encontrados na literatura, mas não existe ainda um consenso sobre qual unidade oferece uma visão de contexto de maior granularidade do que o proporcionado por uma classe de objetos. Este artigo examina três mecanismos citados com freqüência: *subject*, *ensemble* e *subsystem*.

PALAVRAS-CHAVE: Partição de Sistemas, Agrupamento de Objetos, Análise Orientada a Objetos.

1. INTRODUÇÃO

Alguns problemas de software são grandes e complexos para serem compreendidos como um todo. Por essa razão, dividem-se tais problemas em partes menores que sejam mais compreensíveis, estratégia que passou a ser conhecida como "dividir e conquistar", e estabelecem-se as interfaces entre elas para que a funcionalidade total do sistema seja atingida.

BOOCH (1991) considera que o desafio atual consiste no desenvolvimento de aplicações que exibem um conjunto rico de comportamentos, como por exemplo, sistemas que controlam a rota num tráfego aéreo ou que imitam certos aspectos da inteligência humana. Para esses tipos de software, a decomposição é essencial.

Na metodologia estruturada, em essência, decompõe-se o problema em partes e estabelece-se uma representação hierárquica das funções, onde o elemento de maior nível cresce a medida em que se caminha nos sentidos horizontal e vertical da representação (PRESSMAN, 1992).

Uma maneira similar deve ser definida na orientação a objetos (OO), que permita a decomposição de grandes sistemas logo nos estágios iniciais do processo de desenvolvimento. Em geral, os métodos de análise/"design" propõem meios para realizar a decomposição, porém o tema ainda encontra-se em debate.

FICHMAN & KEMERER (1992) consideram que o enfoque "*top-down*" da análise estruturada é preferível em relação à maneira "*bottom-up*" da OO, porque classes e instâncias, mesmo as de mais alto nível, têm menor granularidade e são definidas tarde demais no processo de desenvolvimento para proporcionar uma base de partição. Para MONARCHI & PUHR (1992), enquanto o diagrama de nível micro consiste de objetos e seus interrelacionamentos, não fica claro quais construções devem ser mostradas num diagrama de nível macro, representado pelo agrupamento de classes.

(*) Profª Drª Beatriz M. Daltrini - Professora de Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica - Universidade Estadual de Campinas - FEE/UNICAMP - Caixa Postal 6101 - CEP 13081-970 - Campinas (SP)
(**) Sérgio R. Sigrist - Aluno de mestrado em Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica - Universidade Estadual de Campinas - FEE/UNICAMP - Analista de Sistemas do Centro de Informática na Agricultura da Universidade de São Paulo - USP/Piracicaba - Av. Pádua Dias, 11 - Caixa Postal 9 - CEP 13418-900 - Pircacicaba (SP)

A orientação a objetos evolui rapidamente e novas contribuições são apresentadas. Este artigo examina as caraterísticas das propostas de partição citadas com freqüência na literatura e acrescenta mais detalhes do que os encontrados em FICHMAN & KEMERER (1992) e MONARCHI & PUHR (1992).

2. DEFINIÇÕES

Diversos autores têm se posicionado sobre a importância de se ter um vocabulário comum para discutir a OO. Alguns termos empregados no presente texto serão apresentados a seguir.

Classe é uma coleção de atributos e serviços comuns e ***instância*** é uma ocorrência ou a realização de uma classe. Em geral, o termo ***objeto*** refere-se a ambos, classe e instância (MONARCHI & PUHR, 1992).

Atributos são elementos de dados que descrevem uma instância de um objeto. ***Serviço*** é o processamento realizado por um objeto quando este recebe uma mensagem (COAD & YOURDON, 1990). ***Encapsulamento*** significa a manipulação dos atributos de um objeto unicamente pelos serviços definidos neste mesmo objeto.

Os ***relacionamentos*** mais importantes entre objetos, por serem diretamente representados na implementação, são ***agregação e classificação*** (PITTMAN, 1993). Agregação é usada para modelar como um objeto faz parte de outro objeto, ou como um objeto representa uma coleção de outros objetos. Classificação, ou relacionamento superclasse/subclasse, é realizada através da hierarquia de ***herança***, onde atributos e serviços são transmitidos das classes gerais para as classes específicas.

Outros termos também são definidos na OO. Um estudo aprofundado pode ser encontrado em SNYDER (1993), cujo trabalho resultou num modelo abstrato atualmente adotado pelo Object Management Group, Inc[1].

3. PARTIÇÃO NO PARADIGMA DE OBJETOS

Os elementos de partição encontrados nas abordagens OO de diferentes autores são *ensemble, subject* e *subsystem*.

Ensembles (De CHAMPEAUX et al, 1993) são objetos com propriedades especialmente definidas para forçar uma aproximação com o enfoque "*top-down*" de desenvolvimento.

Subjects (COAD & YOURDON, 1990) são formas de representar um agrupamento de objetos a partir dos relacionamentos de agregação e classificação.

Subsystem (WIRFS-BROCK & JOHNSON, 1990) é um conjunto de classes (e possivelmente outros *subsystems*) que cooperam entre si para satisfazer um conjunto comum de responsabilidades.

Esses três elementos serão mostrados em detalhes a seguir.

3.1. ENSEMBLE

Ensemble (De CHAMPEAUX, 1993) são geralmente grandes encapsulamentos de objetos com propriedades especiais definidas para representar diferentes camadas de abstração e permitir a decomposição "*top-down*".

Ensembles assemelham-se a objetos, pois possuem atributos, podem ter uma máquina de transição de estados associada e podem interagir com outros objetos. Diferem dos objetos pelo fato de agrupar objetos (ou outros *ensembles*) de menor nível denominados *componentes*. Os componentes ficam ocultos de outros objetos e interagem somente com os componentes pertencentes ao mesmo *ensemble*. Desta maneira, um *ensemble* pode ser visto como um *gateway* entre seus componentes e o resto do sistema.

Outra diferença de *ensemble* e objeto é que um objeto pode ser concebido como uma máquina seqüencial, enquanto *ensemble* denota uma entidade com paralelismo interno. Num sistema bancário, uma ***conta*** pode ser um objeto somente quando uma transação for realizada. Por outro lado, a "gerência de empréstimos" pode ser representada como um *ensemble* porque seus componentes, as seções de empréstimo, podem operar em paralelo.

Um *ensemble* oculta os detalhes de seus componentes que são irrelevantes fora do *ensemble*, da mesma maneira que uma linguagem de programação orientada a objeto oculta detalhes internos de implementação dos objetos.

Dois tipos de atributos podem ser definidos: atributos de *ensemble* e atributos de componentes. Atributos de *ensemble* referem-se somente ao *ensemble*, enquanto que atributos de componentes são compartilhados por todos os componentes do ensemble. Por exemplo, uma esquadra, representada como *ensemble*, pode ter os atributos "quantidade de embarcações" (atributo de *ensemble*) e "direção" (atributo compartilhado pelos componentes).

(1) Associação entre empresas para padronizar termos e definições da orientação a objetos.

Na ausência de atributos de componentes, pode-se desenvolver um modelo de transição de estados e introduzir um *trigger* para modelar a interação entre ensemble e componentes. *Trigger* difere de pedido de serviço ou transmissão de mensagem, seu único efeito é promover uma mudança de estado no objeto receptor. Um *ensemble* Esquadra pode receber um aviso "retornar ao porto" e transmiti-lo a todos os seus componentes. Um exemplo de interação entre um *ensemble* e um componente pode ser a Esquadra dando uma direção específica para uma das embarcações.

As propriedades de *ensemble* podem ser resumidas da seguinte maneira:

- *Ensemble* é um objeto cujos componentes funcionais são objetos ou outros *emsembles*.

- Um componente é parte de ,no mínimo, um e, no máximo, um *ensemble*. Essa propriedade garante que o relacionamento ensemble/componente seja não-transitivo e define múltiplas camadas de abstração;

- Um *ensemble* serve como mediador das interações entre os seus componentes e as entidades fora dele;

- Os componentes podem interagir somente entre si e não com objetos fora do *ensemble* do qual fazem parte;

- Um *ensemble* é responsável pela criação e eliminação de seus componentes.

A notação de DE CHAMPEAUX é mostrada na figura 1.

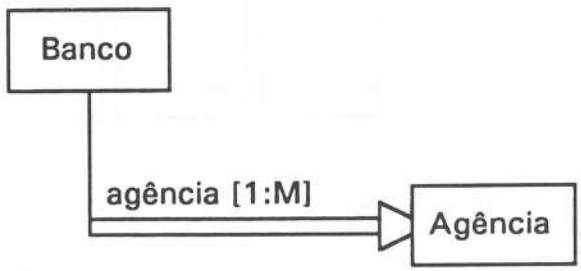


Figura 1 - Exemplo de um *Ensemble* (de Champeaux et al, 1993)

No exemplo tem-se um sistema bancário modelado com o *ensemble*. Os retângulos representam classes. A linha dupla significa que Banco representa um *ensemble* cujos componentes são as agências bancárias.

Os autores propõem artifícios pra realizar a comunicação entre *ensemble* e componente, através da herança múltipla, e expor parcialmente, quando necessário, o comportamento de um particular componente de um *ensemble* ao sistema. Também são propostas formas de modelar um sistema completo a partir de *ensembles*, por meio de adição de atributos especiais nos componentes do *ensembles*.

Uma característica final de *ensemble* é a implementação. Devido às propriedades de encapsulamento e de comunicação, *ensemble* pode ser visto como um objeto e ser manipulado diretamente na implementação.

3.2. SUBJECT

Subject é um mecanismo conceitual definido no método Object-Oriented Analysis (COAD & YOURDON, 1990). Este método propõe realizar um processo de análise em cinco etapas para construir o modelo de objetos:

- Encontrar objetos e classes;

- Identificar estruturas;

- Definir *subject*;

- Definir atributos dos objetos e conexões de instância;

- Definir serviços dos objetos e conexões de mensagem.

Neste processo, *subject* é visto como um meio de facilitar a compreensão de um modelo composto por muitos diagramas. Os autores citam o trabalho de Miller[2], segundo o qual a capacidade de memória de trabalho humana é limitada a "5 até 9 itens ao mesmo tempo". Este resultado pode ser aplicado na OO, colocando-se de 5 a 9 ícones de objetos num desenho.

O ponto de partida para a definição de *subject* são as chamadas estrutura de montagem, que representa o relacionamento de agregação, e estrutura de classificação. Segundo os autores, ambas expressam um método básico de organização do pensamento humano. A estrutura de classificação proporciona uma importante partição do problema, pois distingue agrupamentos de objetos mutuamente exclusivos.

A figura 2 mostra um exemplo da notação de COAD & YOURDON.

(2) MILLER, G. The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review, p. 81-97, Mar. 1956.

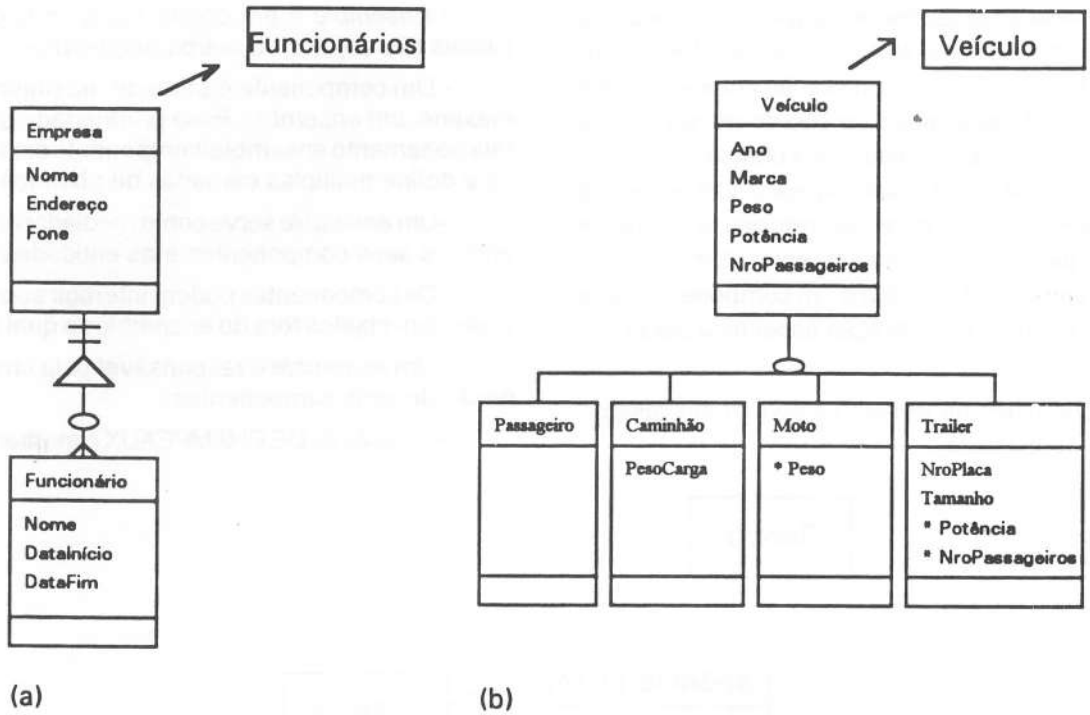


Figura 2 - Estruturas de agregação(a) e classificação (b) (COAD & YOURDON, 1990)

Na figura 2 (a), uma Empresa é mapeada a um conjunto de Funcionários (estrutura de agregação representando coleções). Esta estrutura sugere um *subject* Funcionários, representado pelo retângulo.

Na figura 2 (b), Veículo é uma classe geral e Passageiro, Caminhão, Moto e Trailer são classes especializadas. Esta estrutura pode ser entendida como um *subject* **Veículo**. O mecanismo de herança fica visível. Atributos não herdados pelas classes especializadas são anotados com um asterisco. Novos atributos podem ser alocados às classes especializadas, como indicado na classe especializada Trailer.

Os autores propõem um guia para a identificação de *subject*. Por exemplo, se uma estrutura de classificação apresentar entre 5 e 9 classes, esta estrutura é candidata a *subject*.

A introdução de *subject* está associada à complexidade do modelo, devendo em grandes projetos ser feita logo no início. A recomendação é que uma equipe de analistas faça uma rápida identificação de objetos, estruturas e um conjunto inicial de *subjects*. Este procedimento é denominado "blitz". Tais *subjects* podem ser associados às equipes de projeto e posteriormente refinados.

3.3. SUBSYSTEM

Subsystem (WIRFS-BROCK JOHNSON, 1990) é um conceito definido no método Responsability-Driven Design. O método baseia-se nas responsabilidades e colaborações entre objetos através do modelo cliente-servidor e o mecanismo denominado *contrato*.

As responsabilidades de um objeto são todos os serviços que ele proporciona aos objetos que os solicitam ou os serviços que ele passa a outros (colaboração) objetos. Assim, os objetos assumem as suas responsabilidades executando a sua computação ou colaborando com outros objetos.

Contrato descreve a forma pela qual um cliente pode interagir com o servidor. Ambos devem subscrever um contrato: o cliente apresenta os seus pedidos e o servidor responde apropriadamente aos pedidos. Uma classe pode suportar um ou vários contratos, dependendo se seus serviços forem usados por um ou mais clientes.

Nesse contexto, um *subsystem* significa um conjunto de classes (e possivelmente outros *subsystems*) que cooperam entre si para satisfazer um conjunto comum de responsabilidades. *Subsystems* não são suportados diretamente por linguagens orientadas a objeto, não existindo portanto durante uma execução. Figuram apenas como uma construção da análise. São maneiras de pensar sobre grandes sistemas. Um meio para verificar se um agrupamento de classes forma um *subsystem*

é tentar *dar um nome* a este agrupamento.Se for possível escolher o nome, a função de maior nível que as classes cooperam para encontrar é um *subsystem*.

MARTIN (1993) refere-se a *subsystem* como um *container* de classes. Quando um pedido de Serviço for enviado ao *subsystem*, o pedido é delegado à classe dentro do *subsystem* que suporta o pedido. Olhando de fora, cada *subsystem* assemelha-se a uma classe que suporta seus próprios contratos. Outras classes comunicam-se com o *subsystem* porém não conhecem nada do seu interior. Do mundo exterior, não há diferença entre uma classe, um subsystem ou mesmo uma aplicação total. Tudo são objetos encapsulados com contratos para proporcionar os serviços.

O grafo de colaboração é a ferramenta empregada para representar subsystem, como mostra a figura 3.

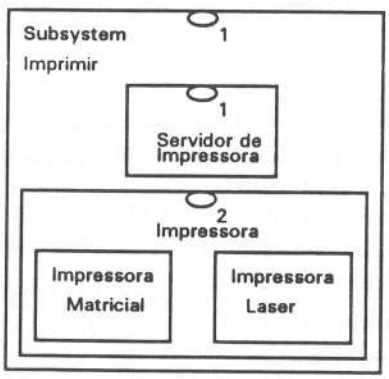


Figura 3 - Grafo de colaboração (WIRFS-BROCK & JOHNSON, 1990)

A figura 3 mostra um *subsystem* denominado Imprimir.Contratos são indicados por semicírculos e classes são representadas por retângulos. O *subsystem* Imprimir encapsula as classes Servidor e Impressora. As subclasses Impressora Matricial e Impressora Laser ficam aninhadas dentro da superclasse Impressora. Todas essas classes colaboram com a impressão de arquivos. Embora a classe Servidor colabore com uma outra classe Fila, esta classe não faz parte do *subsystem* Imprimir porque as instâncias de Fila são usadas por classes fora do *subsystem* Imprimir. Uma classe faz parte de um *subsystem* somente se ela satisfizer a finalidade deste *subsystem*.

Subsystems simplificam a fase de projeto de grandes aplicações na medida em que podem ser tratados como classes. Uma aplicação pode ser decomposta em vários *subsystems* e estes podem ser modelados até que todos os detalhes sejam especificados.

Em razão dos clientes usarem a funcionalidade de um *subsystem* através de um conjunto bem definido de contratos, a funcionalidade de um *subsystem* pode ser ampliada sem romper com o resto da aplicação. Um novo contrato pode ser definido, ou um contrato já existente pode ser ampliado para incluir uma nova funcionalidade. No *subsystem* do exemplo dado, pode-se acrescentar a habilidade de imprimir num determinado instante ou imprimir um numero específico de cópias. Os contratos existentes vão adequadamente lidar com esta nova funcionalidade: o *subsystem* Impressora imprime o arquivo (contrato já existente) mas de diversas maneiras (nova funcionalidade).

4. CONCLUSÃO

Foram mostradas em detalhes as caraterísticas de *ensemble, subject* e *subsystem,* que procuram oferecer uma visão de maior nível do que a representada por uma classe de objetos. A Tabela 1 faz uma síntese dos principais itens descritos no artigo.

Tabela 1 - Síntese de itens descritos

	Ensemble	*Subject*	*Subsystem*
representação	sim	sim	sim
visão de contexto	agregação(c/ restrições)	agregação; classificação	classes funcionais; classificação
guia	sim	sim	sim
comunicação	entre *ensembles; interensemble*	entre *subjects*	entre *subsystems; intersubsystem*
encapsulamento	sim	não	sim
implementável	sim	não	não

Os mecanismos de agrupamento examinados apresentam vários pontos em comum. Todos proporcionam uma ferramenta de representação. Em cada representação fica implícita a visão de contexto. A base é formada por um relacionamento especifico, agregação e classificação, entre as classes pertencentes ao agrupamento. Aqui *subsystem* difere dos demais pois inclui classes que trabalham juntas para obter uma função de mais alto nível.

Todos os mecanismos sugerem um guia para obter a partição. Guias para *subject* (construir uma estrutura de classificação ou de agregação contendo de 5 a 9 classes) e *subsystem* (tenta nomear um agrupamento de classes) são explícitos, enquanto que o guia para derivar um *ensemble* pode ser inferido a partir da definição de ensemble, ou seja, tentar identificar as entidades que decompõem o problema em partes independentes.

Outros conceitos fundamentais da OO são comunicação e encapsulamento. *Ensemble* e *subject* parecem oferecer um suporte mais eficiente. Ao receberem uma mensagem, *ensemble* e *subject* não realizam diretamente o serviço, apenas atuam como "distribuidores" de serviço aos seus elementos. Para utilizar os serviços proporcionados por um ensemble ou subsystem deve-se explicitamente dirigir um pedido ao *ensemble* ou ao *subsystem*, que ocultam os detalhes de implementação de seus elementos internos. Em *subject* não fica claro como os elementos de uma estrutura interagem entre si.

Outra característica importante é a implementação. Enquanto *subsystem* e *subject* são construções da análise, uma representação baseada em *ensemble* é mais robusta pois permite que *ensembles* definidos na análise possam persistir diretamente na implementação.

De uma maneira geral devemos concordar com MONARCHI & PUHR (1992) no sentido de que ainda não existe uma construção (*ensemble, subject, subsystem*, ou outras não analisadas neste artigo) mais indicada para realizar a partição de sistemas. Os guias são heurísticos, não há uma padronização quanto à notação e as unidades de partição são definidas no contexto de um método em particular, portanto deve-se conhecer as sutilezas de cada método. Por exemplo, para usar toda a extensão de *ensemble* deve-se ter familiaridade com máquina de estados, "*trigger*" e herança múltipla.

Ensemble parece ser mais completo e tende a oferecer uma melhor perspectiva ao analista. A definição de propriedades especiais impõe mais rigor, permite criar diferentes camadas de abstração da aplicação e realizar a decomposição logo no início da análise.

REFERÊNCIAS

BOOCH, G. *Object-oriented design with applications*. Redwood, Benjamin/Cummings,1991,580 p.

COAD, P.; YOURDON, D. *Object- oriented analysis*. Englewood Cliffs, Prentice-Hall, 1990,532 p.

DE CHAMPEAUX, D.;LEA, D.; FAURE, P, *Object-oriented system development.* Reading,Addison-Wesley, 1993, 532 p.

FICHMAN, R. G.; KEMERER, C. F. Object-oriented and conventional analysis and design methodologies:comparison and critique. *IEEE Computer*, v 25, nº 10, pp 22-39, Oct, 1992.

MARTIN, J. *Principles of object-oriented analysis and design*. Englewood Cliffs, Prentice-Hall, 1993, 412 p.

MONARCHI, D. E.; PUHR, G. I. A research typology for object-oriented analysis and design. *Communications of ACM*, v 35, nº 9, pp 35-47, Sep., 1992.

PITTMAN, M. Lessons learned in managing object-oriented development, *IEEE Software*, v.10, nº 1,pp 43-53,Jan.,1993.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. 3ed. New York,McGraw-Hill, 1992, 793 p

SNYDER, A. The essence of objects: concepts and terms. *IEEE Software*, v. 10, nº 1, pp 31-42, Jan.,1993.

WIRFS-BROCK, R. J.; JOHNSON, R. E. Surveying current research in object-oriented design. *Communications of ACM*, v. 3, nº 9,pp. 104-124, Sep.,1990.

INTERFACES PARA AMBIENTES DE PROGRAMAÇÃO: PROBLEMAS E PERSPECTIVAS

INTERFACES FOR PROGRAMMING ENVIRONMENTS: PROBLEMS AND PERSPECTIVES

André Luis R. G. CARVALHO*
Profª Drª Heloísa Vieira da ROCHA**

ABSTRACT

Despite the importance normally assigned to programming, it is still very difficult to learn how to write computer programs. There has always been a considerable difficulty in interating with computers. Traditionally, these difficulties are not restricted only to interating in programming enviromments. However, in other domains, one can presently observe significant changes in these aspects. These changes can be seen as a very positive response to the increasing improvement in the man-machine interaction style that has been occuring lately in these domains. Despite of the many currently, available programming tools, programming is still an activity that is essencially done in an old fashioned way. Recent publications show that for many researchers that is a very worth-studying issue and reveal many inovative proposals. These proposals are based on Artificial Intelligence (AI) research results an are related mainly with the nature of the programming activity, problem solving and human interaction in computer environments. A prelliminary observation of these proposals can show clearly that a revolution is occuring, not only in the programming environments interaction style, byt also in the paradigms that support them, causing significant changes in what is meant by programming. The objective of this article is to explore a bit these issues, allowing this way further and deeper analysis of the several aspects concerned to it.

RESUMO

Apesar da enorme importância que é atribuida à atividade de programar, aprender a programar ainda é uma tarefa muito difícil. Embora essa dificuldade de apropriação tradicionalmente nunca tenha sido restrita ao âmbito dos ambientes de programação, na atualidade podem-se observar conquistas muito significativas em outros domínios da computação, frutos de uma maior preocupação com o estilo de interação a ser mantido com o usuário. Apesar de se poder observar a disponibilização de muitas ferramentas de apoio à programação, programar ainda é uma atividade que se faz à moda antiga. Literatura recente mostra que essa questão tem constituído preocupação de muitos pesquisadores, e que propostas bastante inovadoras têm surgido. Essas propostas são baseadas em resultados de pesquisas em Inteligência Artificial (IA) quanto à natureza da atividade de programação, resolução de problemas e interação com ambientes de computação. A observação preliminar dessas propostas evidencia uma revolução não somente no estilo de interação com ambientes de programação, mas também nos paradigmas que os suportam, mudando significamente o que se entende por programar. O objetivo deste trabalho é trazer à tona toda essa problemática, de modo a servir de subsídio para um trabalho de análise dos diversos aspectos envolvidos na questão.

(*) Pesquisador do Centro de Pesquisa e Desenvolvimento da TELEBRÁS - Professor do I.I. - PUCCAMP e do CTC - UNICAMP. Mestrando em Ciência da Computação - DCC - IMECC - UNICAMP
(**) Professora do Departamento de Ciência da Computação - DCC - IMECC - UNICAMP. Pesquisadora do Núcleo de Informática Aplicada à Educação - NIED - UNICAMP.

1. INTRODUÇÃO

Todos sabemos da extrema importância da atividade de programar computadores, que não só constitui meio rico para formalização e depuração de conceitos e idéias sobre resolução de problemas, mas que em última instância, representa a única maneira possível de se ter controle pleno sobre a máquina.

Em contrapartida, qualquer um que tenha passado pela experiência de ensinar programação, pode dizer sobre quão árduo é o processo de aprendizagem, sobre quão íngreme e tortuoso é o caminho a ser percorrido. Mas afinal, porque é tão difícil aprender a programar?

Se deixarmos de lado, por um instante, o domínio da programação, e nos lembrarmos um pouco de como eram, há algum tempo atrás, as coisas neste aspecto no domínio dos aplicativos, certamente nos recordaremos da aversão pré-concebida que sempre existiu, por parte das pessoas, à simples idéia de ter que interagir com um sistema de computação.

Podemos observar que isto andou mudando e, que hoje, já não mais se verifica de forma generalizada aquela rejeição que outrora percebíamos. Naturalmente não era gratuito o estabelecimento daquela situação, assim como certamente também não o são, as mudanças que ora podemos ali verificar, o que nos leva a crer - e agora voltamos a nos lembrar de que falávamos de programação - que a trajetória evolutiva por que passou o domínio dos aplicativos tem muito para nos ensinar.

O que se verificou no domínio dos aplicativos, foi o desenvolvimento de novas formas de interação homem-computador, o que resultou em *interfaces* modernas, das quais as pessoas se apoderam com extrema facilidade. O mesmo não se deu no domínio da programação, onde poucas mudanças são percebidas, e com o qual, ainda hoje, se interage de maneira essencialmente textual.

Um exemplo ilustrativo dessa problemática, é a linguagem Logo. A linguagem Logo é fortemente calcada em objetivos educacionais (no sentido de ser uma linguagem fácil de ser aprendida), e sua "porta de entrada" tradicionalmente sempre foi seu ambiente gráfico de programação, aliado à exploração de atividades espaciais. Essencialmente, através de não mais que uma dezena de comandos e da estrutura de procedimentos, resultados gráficos extremamente interessantes são conseguidos muito rapidamente.

Se outrora suas capacidades gráficas constituíram seu principal atrativo, hoje em dia, com o surgimento de aplicativos gráficos sofisticados, com *interfaces* de última geração, muito fáceis de serem aprendidos, e que produzem resultados muito atraentes também muito rapidamente, Logo corre o risco de perder seu "carro chefe".

Tendo em vista a já discutida importância da atividade de programar, simplesmente substituir o ambiente gráfico Logo por um sofisticado editor não constitui solução aceitável. Acredita-se na necessidade de buscar novas alternativas de *interfaces* para programação, e pode-se observar em literatura recente, que esta é a preocupação de um grande número de pesquisadores.

Propostas bastante inovadoras têm sido apresentadas. De uma análise preliminar dessas propostas, podemos observar que não só no estilo de interação são necessárias mudanças, mas que aliadas a estas, surgem também novos paradigmas de programação, mudando significativamente o que se entende por atividade de programação.

Portanto, a proposta de uma nova *interface* para um ambiente de programação deve estar calcada em três estudos básicos: nas mudanças na forma de interação com o computador, nos objetivos da atividade de programar no ambiente a ser alterado, e na análise das novas propostas e paradigmas de programação que vêm surgindo como resultado de pesquisas recentes.

2. O DOMÍNIO DOS APLICATIVOS

Há bem pouco tempo atrás, não somente aprender a programar acontecia de maneira traumática. As pessoas tinham uma resistência muito grande ao uso de programas de computador de uma forma geral. Quando interrogadas sobre as razões dessa resistência, as pessoas se justificavam dizendo que não se sentiam capazes de aprender a lidar com algo "tão complicado", que se sentiam inseguras, e que tinham muito medo de cometer erros.

Certamente não podemos culpá-las. Isso reflete claramente não só o hermetismo da área, os tabus que a envolvem, as idéias mistificadas que as pessoas têm dela, mas também, e principalmente, a pouca atenção que sempre se deu à questão da qualidade da interação entre as pessoas e os sistemas de computação.

Entretanto, nos últimos anos, muitos esforços têm sido feitos no sentido de melhorar a qualidade da interação entre o homem e o computador. Podemos ver resultados desses esforços sendo colocados no mercado o tempo todo pelos fabricantes de *software*, que de certo modo, estão aprendendo a respeitar cognitivamente o ser humano que interage com seus produtos.

Assim, vimos surgirem os *menus* de função em substituição à escrita de comandos e,sempre que possível, à especificação dos dados;vimos surgirem as interfaces gráficas, onde o usuário deixa de caminhar por um *menu* em busca de opções, e passa a apontar o que deseja, e onde formas icônicas substituem a interação

textual; tudo isto levando os programas a exigirem menos da memória de seus usuários, possibilitando a ocorrência de poucos erros, e fazendo com que os usuários se sintam mais confortáveis e confiantes.

No entanto, à medida que mais poder vai-se conferindo aos programas, também mais complexos eles vão se tornando. As aplicações vão passando a englobar tantas facilidades, que se torna praticamente impossível conhecê-las e empregá-las todas. Muitos usuários se sentem confusos e, de modo geral, não conseguem dominar por completo o produto que utilizam. Apesar disso, com toda a certeza, continuarão sempre a sentir falta desta ou daquela facilidade ainda não contemplada no produto.

Por um lado, parece possível convencer-se de que não é possível fechar um conjunto de facilidades que venha a atender plenamente a todas as necessidades dos usuários; por outro lado, vemos que na tentativa de se aproximar desta cobertura total, acaba-se tendo aplicações com *interfaces* carregadas e difíceis de serem aprendidas.

Um problema de que sofre praticamente todo sistema de computação é que suas *interfaces* não são modificáveis ou ampliáveis, mesmo nas mãos de usuários experientes e que potencialmente poderiam fazê-lo (Baecker 1987). Esta limitação acaba por fadar as aplicações, mesmo aquelas que possuem uma *interface* fácil de ser aprendida e usada, a eventualmente se tornarem incômodas. Isto porque elas não crescem com seus usuários, e nem podem facilmente ser adaptadas às necessidades individuais destes.

Pode-se observar que na tentativa de minimizar o impacto desse problema, vem se tornando mais corriqueiro o uso de um recurso relativamente antigo conhecido por *macro*. Este recurso tem sido empregado nas *interfaces* modernas da seguinte maneira: o usuário coloca a aplicação em um modo de operação no qual ela registra todas as ações do usuário e as unifica em uma seqüência identificável e passível de ser ativada posteriormente pelo usuário.

Apesar de efetivamente representarem algum progresso no sentido da extensibilidade da *interface, macros* são extremamente frágeis (Lieberman 1992), pois são limitadas a repetir exatamente a seqüência de operações registrada pela aplicação, sendo via de regra muito sensíveis a detalhes irrelevantes do contexto em que foram definidas.

Existe um potencial muito grande no sentido de uma nova classe de aplicações, mais expressiva, mais ampliável e que respeite mais a imaginação do usuário (Eisenberg 1991). A idéia central consiste em se ter nas aplicações, tanto *interfaces* de manipulação direta com capacidade de aprendizado, como um interpretador para pelo menos uma linguagem de programação.

Com isto, os usuários passariam a ter acesso a certos elementos de linguagens de programação que normalmente estão ausentes das *interfaces* que atualmente existem.

Um exemplo de aplicação destinada a usuários finais que fez uma tentativa neste sentido, é o Guide 3.1. Trata-se de um sistema de autoria de *hipermidia*; foi o primeiro disponível para microcomputadores IBM e Macintosh. Um documento do Guide é chamado *guideline*, e é uma mistura de texto e gráficos. Certas palavras, frases, ou objetos gráficos, podem funcionar como botões que provêm acesso a *guidelines* subjacentes. Existem diversos tipos de botão, cada um com uma funcionalidade diferente, e.g., botões de expansão, de nota, de referência e de comando.

Botões de comando são justamente a porta de acesso à programação. Este tipo de botão deve ser associado a um programa escrito em uma linguagem de programação própria, de alto nível, bastante poderosa e muito semelhante a Pascal. O pressionamento de um botão deste tipo, tem como efeito a execução do programa que lhe foi associado.

Como dissemos acima, a linguagem usada para escrever programas é bastante semelhante a Pascal, ou seja, trata-se de uma linguagem textual. Ao construir aplicações com esta ferramenta, podemos sentir claramente a quebra de contexto que acontece, quando do ambiente de autoria de *hipermidia*, passamos para um editor de textos, a fim de escrever o texto do programa a ser associado a um botão de comando do hiperdocumento em que trabalhamos.

Enquanto nas *interfaces* modernas o usuário interage com seus objetos de interesse, num ambiente gráfico de manipulação direta, para ampliar as funcionalidades da *interface*, normalmente é requerido do usuário o domínio de uma linguagem de programação convencional, como C ou Lisp, por exemplo. A distância que existe entre estes dois ambientes constitui uma espécie de "Muro de Berlim" (Lieberman 1992), impedindo que o usuário tenha controle pleno sobre suas aplicações.

É importante ressaltar que nos exemplos que temos atualmente, ao inserirmos uma linguagem de programação no domínio dos aplicativos, com ela trazemos também toda problemática que existe no domínio da programação e sobre a qual já comentamos anteriormente.

3. O DOMÍNIO DA PROGRAMAÇÃO

Como vimos, no que diz respeito à deselitização do uso do computador, são notórias as conquistas que têm ocorrido no domínio dos aplicativos, o que nos mostra claramente que as mudanças de estilo de interação ali

promovidas são um estrondoso sucesso, e nos convida à reflexão e à análise. Surge assim o interesse de procurar promover modificações semelhantes na forma atual de interação no domínio da programação.

Sem dúvida nos ambientes de programação têm ocorrido inovações, e hoje dispomos de muitas facilidades das quais há bem pouco tempo atrás não dispunhamos. São editores sensíveis à sintaxe, que vão nos advertindo sobre construções mal elaboradas à medida que vamos introduzindo nossos programas.

São ambientes integrados de edição, compilação e execução, que apontam erros de sintaxe e de execução, permitem que sigamos nossos programas diretamente no código fonte, e que fazem a ligação de código de biblioteca automaticamente, sem que precisemos nos preocupar em especificar de que bibliotecas precisamos ou suas localizações.

São sistemas de *help on-line* sofisticados, sensíveis ao contexto e baseados em tecnologia de hipertexto, que permitem diversas formas de acesso à informação, fornecendo ajuda descritiva, procedimental, interpretativa, navegacional e orientada por objetivo, inclusive dando exemplos que podem ser cortados e colados na área de trabalho.

São linguagens não procedimentais, onde descrevemos não o procedimento a ser seguido para a obtenção de um certo resultado, mas sim o resultado que se visa obter, e a ferramenta produz o resultado desejado, seja pela interpretação direta da especificação, seja pela geração de código que o faça. Ferramentas deste tipo existem para alguns domínios de aplicação, e. g., consultas a bancos de dados, geração de relatórios e geração de *interfaces* homem-computador.

Enfim, é toda uma gama de ferramentas sofisticadas que embora facilitem a tarefa de desenvolver programas, se limitam a ajudar na depuração, em prover um acesso imediato e rápido a informações de manual ou a diminuir o volume de programação, mas que não promovem nenhuma inovação no que diz respeito ao estilo de interação com a atividade de programar. Assim, apesar da modernidade que podemos identificar neste domínio, e em geral em todas as áreas de aplicação, escrever programas ainda é uma tarefa que se faz à moda antiga.

Achamos importante que fique claro que quando falamos em mudanças, não nos referimos a esse tipo de mudanças que mencionamos acima, e que podemos observar no domínio da programação atualmente. Precisamos sim, de mudanças que façam uso qualitativo da moderna tecnologia de *interfaces*, no sentido de facilitar a apropriação por parte do usuário.

Um exemplo ilustrativo desta problemática é a linguagem Logo. A linguagem Logo foi desenvolvida por volta de 1968, e a idéia de seu projeto surgiu inicialmente com o propósito de criar uma linguagem que pudesse vir a substituir o Basic.

Assim, foi concebida para ser uma linguagem poderosa, não apenas com capacidade de processar listas e de permitir a criação de novos procedimentos, caraterísticas que não eram exibidas pelo Basic, mas fundamentalmente, que fosse de fácil aprendizado.

Originalmente, Logo não dispunha de facilidades gráficas, já que os computadores da época não possuíam capacidades desta natureza. Seymour Papert, através de intensa utilização da linguagem, e de inúmeras pesquisas, conseguiu dar ao Logo uma nova roupagem e uma estrutura filosófica (Valente 1991).

É universalmente sabido que Logo é uma linguagem fortemente calcada em objetivos educacionais, o que a leva a ser amplamente utilizada não só para introduzir programação em cursos de informática dos mais diversos níveis, mas também, e principalmente, com crianças, com o intuito de estudar os impactos da tecnologia da programação sobre o desenvolvimento cognitivo.

Suas capacidades gráficas, utilizadas principalmente para explorar atividades espaciais, historicamente têm constituído a "porta de entrada" para o aprendizado de programação, através de um contato imediato do aprendiz com o computador.

Se outrora essas capacidades representaram o principal atrativo da linguagem, hoje em dia, com o advento das *interfaces* gráficas, e de sua exploração por inúmeras aplicações sofisticadas e de fácil inserção, isso tem deixado de ser verdade, movendo o foco de interesse dos aprendizes na direção de editores gráficos e de outras aplicações do gênero, fazendo assim com que Logo perca seu "carro chefe".

Somos forçados a concordar que o "conforto" da interação provida por essas aplicações não se compara àquele do ambiente Logo, onde para obter qualquer resultado, por mais simples que seja, é preciso escrever código de programação. Por outro lado, sabemos da extrema importância da atividade de programar. Assim, nos parece claro que o atual estilo de interação com o ambiente Logo precisa ser repensado, posto que se mostra antigo e ultrapassado, o que pode tornar desmotivante seu uso.

Buscar respostas para a problemática da interação com as linguagens de programação constituem presentemente preocupação de muitos pesquisadores, que vêm apresentando propostas inovadoras para o encaminhamento da solução desta problemática.

Da observação dessas propostas, pode-se verificar que a efetividade de uma proposta está ligada a três grandes aspectos: ao domínio da programação propria-

mente dito, à tecnologia de *interfaces* homem-computador, e ao suporte de paradigmas de programação adequados. Assim, surgem com elas, não somente mudanças na forma de interagir com o domínio da programação, como também novos paradigmas de programação.

O exame de algumas dessas novas propostas e paradigmas, nos dá uma idéia do fervilhar que ocorre presentemente na área, o que coloca em clara evidência a necessidade e o interesse de pesquisa existente na mesma.

3.1. NOVOS PARADIGMAS DE PROGRAMAÇÃO

Papert coloca que os paradigmas de programação estruturam a atividade de programar, e que implicam na maneira pela qual o programador pensa sobre essa tarefa (Baranauskas 1993).

Assim, parece natural que, em resposta à demanda por mudanças no domínio da programação, surjam novos paradigmas de programação, trazendo com eles novas formas de estruturar a atividade de programar, e em última instância, novas concepções acerca desta tarefa.

3.1.1. ORIENTAÇÃO A OBJETOS

O paradigma de orientação a objetos já é um paradigma relativamente antigo, sendo hoje praticamente considerado um clássico e seu uso vem aumentando com o tempo.

Esse paradigma propõe que os programas simulem o mundo real. Assim, posto que o mundo real é constituído nem por dados, nem por processos, mas sim por coisas que o paradigma chama de objetos, propõe que estes sejam os elementos constituintes dos programas.

O paradigma envolve dois aspectos, um modelo de computação e uma filosofia de desenvolvimento.

O modelo de computação proposto, prevê um mundo todo constituído apenas por objetos, que trabalham e cooperam entre si através da troca de mensagens.

Todo objeto possui um estado interno, que é representado em uma memória local inacessível e indevassável por outros objetos. Além de um estado interno, todo objeto possui também um comportamento, que é representado por um repertório de ações de que o objeto dispõe a fim de responder a demandas (mensagens) externas ou mesmo internas ao próprio objeto.

No modelo, todo processamento é ativado pela troca de mensagens, e todo objeto, ao receber uma mensagem, passa a executar uma ação específica em reação à mensagem recebida, que poderá resultar em alterações em seu estado interno, no envio de novas mensagens, e mesmo na criação de novos objetos. Assim, um programa pode ser visto como uma sociedade de objetos que interagem trocando mensagens, e que reagem em resposta a mensagens recebidas.

A filosofia de desenvolvimento proposta prevê que os objetos tendem a exibir facetas similares, o que nos permite agrupá-los em classes, de modo a fatorar propriedades a serem herdadas por todos seus irmãos de classe. Analogamente, classes podem ser agrupadas em superclasses, etc. Assim, desenvolvimento orientado por objetos resume-se na identificação de objetos e no agrupamento deles em classes (Takahashi 1988).

Dentre os ambientes que suportam o paradigma, podemos destacar: o SmallTalk, que é considerado um ambiente clássico de orientação a objetos; os diversos ambientes C++, que são inúmeros hoje em dia; o Visual Basic, que valida o paradigma de orientação a objetos ao moderno paradigma da programação visual, sobre o qual em seguida falaremos um pouco mais, e por fim ambientes como o Turbo Pascal, que tradicionalmente não foram projetados para seguir esse paradigma, e que hoje, em suas versões mais modernas, passam a incorporá-lo. Mesmo para a linguagem Logo tem-se o Object Logo da Aplle e MicroWorlds da Logo Computer Systems Inc., que implementam orientação a objetos.

3.1.2. ORIENTAÇÃO A EVENTOS

O paradigma de orientação a eventos é um dos paradigmas que surgiram à sombra do paradigma de orientação a objetos. Entende-se por evento qualquer ação reconhecida por um objeto e para o qual se pode escrever código de tratamento.

Eventos podem ter origem em ações de usuário, podem ser provocados via programação, ou ainda, podem ser disparados pelo sistema. Assim, tudo se passa como se a sociedade dos objetos se mantivesse permanentemente à espreita, à espera da ocorrência de eventos que os sensibilizem. Quando isso ocorre, o objeto dispara uma reação em resposta ao evento e volta ao estado inicial.

Trata-se de um paradigma bastante adequado para modelar uma série grande de tipos de problemas, em especial, problemas com caraterísticas de tempo real. Tem sido sobremaneira usado em sistemas gerenciadores de *interface* homem-computador, onde os objetos são os elementos de *interface*, e.g., botões, caixas de texto, etc. Dentre os sistemas que seguem este paradigma, podemos destacar o MS Windows e o X Windows, ambos sistemas gerenciadores de *interface*.

3.1.3. PROGRAMAÇÃO VISUAL

O paradigma da programação visual também é um dos paradigmas que surgiram à sombra do paradigma de orientação a objetos. Ele pressupõe a existência de uma *interface* de manipulação direta com capacidade de manipulação de uma série de objetos com aparência e comportamento pré-definidos e/ou configuráveis, que possam ser acoplados na construção de objetos de mais alto nível.

Tem sido extensivamente usado em sistemas geradores de *interface* homem-computador, onde os objetos são os elementos de *interface*, e.g., botões, caixas de texto, etc. Seu uso neste contexto é bastante natural, posto que elementos de *interface* são de natureza totalmente visual, e é ideal que possuam aparência e comportamento pré-definidos e configuráveis.

Nesse contexto, o paradigma é normalmente usado aliado ao paradigma de orientação a eventos, que como já dissemos, é muito usado como base de sistemas gerenciadores de *interface*, que costumam ambientar não só esse tipo de aplicativo, mas ainda os sistemas cuja *interface* é gerada por eles. Dentre os sistemas que seguem este paradigma, podemos destacar o Visual Basic e o Visual C++ para ambiente PC, e o TeleUSE em ambiente de estação de trabalho, todos sistemas geradores de *interface*.

3.1.4. PROGRAMAÇÃO POR EXEMPLOS

O paradigma de programação por exemplos constitui uma inovação mais radical. A tônica deixa de ser uma forma a ser empregada na escrita de programas, posto que não mais se configura a atividade de escrever programas.

Assim, ao programar o computador utiliza-se a metáfora de "ensinar" tarefas ao computador, o que se dá através de exemplos e contra-exemplos. A interação se estabelece da seguinte forma: o usuário apresenta ao computador uma série de instâncias da tarefa que lhe deseja "ensinar", e este abstrai dos exemplos os procedimentos necessários para a realização da tarefa desejada (Bauer 1979).

Não dispomos comercialmente de nenhum ambiente de programação de propósito geral que suporte este paradigma. O que existe, são diversos protótipos de pesquisa, incluindo propostas de aplicações como editores de texto, ou editores gráficos, e que incorporam características de aprendizado através de exemplos em suas *interfaces*.

3.2. NOVAS PROPOSTAS PARA AMBIENTES DE PROGRAMAÇÃO

3.2.1. BOXER

Programação é quase sempre vista como um meio que os especialistas têm de fazer com que os computadores realizem tarefas complicadas de modo eficiente e confiável. Boxer propõe uma visão alternativa: programação como uma atividade cotidiana para a maioria das pessoas; programação como uma forma de usuários não especialistas controlarem um meio reconstrutível.

Um meio reconstrutível deve não somente servir para especialistas desenvolverem produtos profissionais, mas também atender às necessidades de pessoas comuns, e.g., crianças, professores, e outros usuários de computador não especialistas, e nele, qualquer usuário, sem distinção de nível de perícia, deve ter acesso às mesmas ferramentas de trabalho.

O meio reconstrutível deve ser programável e aberto, o que implica que mesmo produtos produzidos profissionalmente passam a ser mutáveis, adaptáveis, fragmentáveis, e compartilháveis. Assim, não somente profissionais são capazes de construir seus produtos, nele, mas qualquer um é capaz de reconstruir versões personalizadas destes mesmos produtos, o que estabelece uma situação bastante contrastante com a situação atual dos aplicativos de *software*.

Desse modo, necessidades e restrições tradicionalmente impostas às linguagens de programação, e.g., simplicidade formal, eficiência, verificabilidade, e uniformidade convivem com outras imposições, tais como ser de fácil compreensão, parecer familiar, ser expressiva, e ser extremamente interativa.

Assim, Boxer baseia-se em dois princípios fundamentais: metáfora espacial e realismo ingênuo. Todos os objetos computacionais são representados em termos de caixas, que são regiões da tela que podem conter texto, gráficos e outras caixas (o que estabelece uma estrutura de hierarquia).

Dessa forma, uma variável é uma caixa que contém seu valor; estruturas de dados compostas são caixas que contém outras caixas (como uma variável que contém outras variáveis); um programa é uma caixa que contém o texto do programa; e subprogramas e variáveis são representados como sub-caixas.

Realismo ingênuo é uma extensão do conceito *what you see is what you have*. Assim, o usuário pode pensar que o que ele vê na tela é seu mundo computacional inteiro, e.g., (1) todo texto que aparece na tela, seja ele produzido pelo sistema, entrado pelo usuário, produzido

por um programa, pode ser movido, copiado, modificado, ou, no caso do texto ser um programa, executado; (2) pode-se alterar o valor de uma variável simplesmente alterando o conteúdo de sua caixa na tela, e se uma variável tem seu valor atualizado por outros meios, o conteúdo de sua caixa automaticamente espelha isto.

Boxer é uma extensão da linguagem Logo; pequenos procedimentos Boxer, especialmente aqueles que trabalham com gráficos, se parecem muito com procedimentos Logo (diSessa 1986).

Podemos observar no Boxer o uso do paradigma de programação visual aliado ao paradigma de programação procedimental. O paradigma visual é usado na elaboração da estrutura hierárquica da aplicação, que se dá através do aninhamento de caixas; e o paradigma procedimental é usado para escrever texto de código para caixas que representam ações.

Existem versões comerciais de Boxer para computadores da linha Macintosh, da linha PC e também para estações de trabalho SUN.

3.2.2. MONDRIAN

Mondrian é um editor gráfico orientado a objetos com capacidade de "aprender" novos procedimentos gráficos através do paradigma de programação por exemplos.

O usuário pode demonstrar uma seqüência de comandos de edição gráfica através de um exemplo concreto com o propósito de ilustrar o funcionamento esperado do novo procedimento.

Um agente da *interface* grava os passos do procedimento em forma simbólica, usando técnicas de aprendizagem do domínio da Inteligência Artificial (IA), observando relacionamentos entre objetos gráficos e dependências entre operações da *interface*.

O agente generaliza então um programa que pode ser usado para resolver exemplos análogos. A heurística de generalização não é comparável às tradicionais *macros*, que se limitam apenas a repetir a exata seqüência de passos que a definiu.

Como já dissemos, o sistema utiliza o paradigma da programação por exemplos, e representa internamente todas as operações através de seqüências pictóricas de exemplos, aliando assim o poder da representação procedimental e facilidade de uso das *interfaces* gráficas. (Lieberman 1992)

3.2.3. METAMOUSE

Metamouse é um ambiente de edição gráfica onde um agente ajuda o usuário através da automatização de tarefas de edição repetitivas. O objetivo do projeto foi tornar a tarefa de programar o mais parecida possível com a tarefa de editar, com uma quantidade mínima de interação extra, com a finalidade de dirimir situações de ambigüidade.

Para especificar um procedimento o usuário constrói um exemplo, provavelmente desenhando construções gráficas, de modo a tornar as relações explícitas. O sistema generaliza então esta seqüência em um programa com variáveis. Quando o usuário repete parte da seqüência, Metamouse prevê e realiza automaticamente o restante da mesma.

O usuário pode introduzir novas ações, que tomarão a forma de ramos condicionais. O usuário pode também corrigir erros no programa através da demonstração do exemplo correto ou indicando ícones que mostrem inferências alternativas.

O usuário expressa suas intenções através da metáfora de "ensinar" uma tartaruga gráfica chamada Basil. O uso deste tipo de metáfora foi que motivou o uso de um agente e de técnicas de aprendizado incremental.

O sistema faz uso do paradigma de programação por exemplos, aliado a uma *interface* de manipulação direta. (Maulsby 1993)

3.2.4. MICROWORLDS

MicroWorlds é a mais nova versão do ambiente Logo e se encontra disponível para computadores Macintosh. O estilo de interação com sua *interface* é totalmente gráfico, de conformidade com o padrão Macintosh.

Um projeto feito neste ambiente é organizado em páginas, e nelas pode-se utilizar recursos como desenho, animação, som. etc.

Desenhos podem ser feitos através do Centro de Desenhos, onde se dispõe de recursos no estilo dos editores gráficos modernos, como o PaintBrush por exemplo. Assim, temos facilidades para desenhar formas geométricas (linhas, retângulos sólidos e vasados, elipses sólidas e vasadas, etc), para fazer desenhos a mão livre, para colorir (palheta de seleção de cores, pintura e *spray*). etc.

Pode-se realizar programação de cores, i.e., programar ações a serem disparadas quando for pressionado o botão do *mouse* com o cursor apontando uma determinada cor.

É possível se ter várias tartarugas simultaneamente em uma página, sendo permitido criar novas tartarugas e remover tartarugas dinamicamente. Toda tartaruga tem um nome, e é possível através dele endereçar comandos a elas. Tartarugas podem assumir não só a

forma de tartaruga, mas ainda diversas outras formas. Para se lidar com formas, dispõe-se do Centro de Formas, onde é possível escolher uma forma para a tartaruga, estampar a forma de uma tartaruga na página como um carimbo, editar novas formas, etc.

Pode-se fazer com que tartarugas se comportem como se fossem botões, i.e., pode-se fazer com que seja disparado um conjunto de ações quando do pressionamento do botão do *mouse* com o cursor apontando uma delas. Para tanto, devem-se programar as ações desejadas no quadro de diálogo que possui toda tartaruga. Desta forma, podem-se construir animações programando o quadro de diálogo de uma tartaruga para que ela continuamente se mova e/ou troque de forma.

Além da possibilidade de fazer com que tartarugas se comportem como se fossem botões, dispõe-se também de botões propriamente ditos, e ainda de outros elementos de *interface*, como por exemplo, caixas de texto, molduras, mecanismos deslizantes de controle, etc.

Outra ferramenta disponível no ambiente é o Editor de Melodias. Nele pode-se editar uma melodia fazendo uso dos seguintes recursos virtuais: teclado, controlador de volume, controlador de tempo de notas e pausa, e palheta de instrumentos (onde pode-se escolher o instrumento que executará a melodia editada - violino, clarineta, etc). As melodias editadas podem ser gravadas sob algum nome, e executadas tanto no próprio editor, como no decorrer de um programa, através de instruções específicas.

A todo projeto pode ser associado um programa, que pode ser escrito na página de procedimentos que todo projeto tem. Além disto, toda página pode ser impressa, e ainda pode-se importar e exportar figuras. Por fim, todos os recursos de um projeto, tais como páginas, melodias, procedimentos e sons, podem ser compartilhados por diversos projetos. ("MicroWorlds" 1993)

O sistema faz uso dos paradigmas procedimental e funcional, que suportam a linguagem Logo, bem como do paradigma de orientação a eventos a nível de *interface*. Ainda, toda interação com o sistema se dá através de uma *interface* gráfica de manipulação direta.

4. CONCLUSÃO

Apesar da enorme riqueza intrinsecamente inerente à atividade de programar, essa atividade tem sido reservada para "iniciados", e por isso, ou para isso, a despeito de já há algum tempo estar sendo feito muito esforço no sentido de melhorar a qualidade da interação com os usuários de sistema de computação, e conseqüentemente, no sentido de tornar a computação acessível a um maior número de pessoas, a mesma preocupação tradicionalmente não se observa no domínio dos ambientes de programação.

Pode-se observar resultados significativos destes esforços em praticamente todas as áreas da computação, o que culmina numa crescente deselitização da computação e conseqüentemente em sua penetração em praticamente todos os segmentos da atividade humana.

Haja vista ser o único meio que possibilita exploração total dos recursos da máquina e controle completo sobre a mesma, a importância da atividade de programar tem sido resgatada. Ambientes de programação são então trazidos para o universo dos não "iniciados", seja simplesmente como um ambiente de programação em sua forma mais pura, seja embutido na *interface* de programas aplicativos.

A aliança desses dois domínios se configura muito poderosa, mas existem "efeitos colaterais". Os ambientes de programação adentram o universo dos não especialistas e trazem consigo todos os problemas não resolvidos em seu domínio. Assim, apesar de sua disponibilidade para usuários finais, sua apropriação não ocorre.

Na pequena amostra apresentada neste texto, as propostas que vêm surgindo apontam para muitos sentidos diferentes. Faz-se mister colecionar, analisar e compilar as propostas que vêm surgindo, de modo a identificá-las e a lhes apreender os fundamentos, a aplicabilidade e a abrangência, gerando assim a cultura necessária à obtenção de resultados.

Concluindo, a efetiva deselitização da atividade de programar exige mudanças importantes nesta atividade. Esta questão tem sido do interesse de muitos pesquisadores, que têm feito propostas inovadoras no sentido de tornar programação acessível em âmbito mais geral.

Pode-se observar que essas propostas se apóiam significativamente em resultados de IA com relação aos mecanismos de aprendizagem e aos aspectos cognitivos envolvidos na atividade de programar.

O que se objetiva é empregar as modernas tecnologias de projeto de *interfaces* nos ambientes de programação, de modo a minimizar o esforço mental requerido do usuário na aprendizagem da atividade de programação e no desempenho dessa atividade.

REFERÊNCIAS

(Bae87) Baecker,R. M.: and Buxton,W. A. S., "Research Frontiers and Unsolved Problems" in **Readings in Human-Computer Interaction: A Multidisciplinary**

Approach. Baecker, R. M.; and Buxton W.S. eds., Morgan Kaufmann, California, 1987.

(Bar93) Baranauskas, M. C. C., "Procedimento, Função, Objeto ou Lógica? Linguagens de Programação vistas pelos seus paradigmas" in **Computadores e Conhecimento - Repensando a Educação**. Valente, J. A. org., NIED-UNICAMP, Campinas 1993.

(Bau79) Bauer, M. A., "Programming by Example" in **Artificial Intelligence**, 12:1, May, 1979.

(dA86) diSessa, A. A.,and Abelson, H., "Boxer: A Reconstructible Computational Medium" in **Communications of the ACM**, 9:29, September, 1986.

(Eis91) Eisenberg, M., "Programmable Applications: Interpreter meets Interface".MIT Laboratory for Computer Science, 1991.

(LCS93) "MicroWorlds How To". Logo Computer Systems Inc., 1993.

(Lie92) Lieberman, H., "Mondrian :A Teachable Graphical Editor" in **Visible Language Workshop**. MIT Media Laboratory 1992.

(Mau93) Maulsby, D., and Witten, I. H., "Metamouse: An Instructible Agent for Programming by Demonstration" in **Watch What I Do: Programming by Demonstration**. , A.; Halbert, D.C. ...(et al) eds. The MIT Press, Cambridge, Massachussets; London, England, 1993.

(Tak88) Takahashi, T., "Introdução à Programação Orientada a Objetos", III EBAI-Escola Brasileiro-Argentina de Informática, 1988.

(Val91) Valente,J. A., Logo: mais do que uma linguagem de programação" in **Liberando a Mente - Computadores na Educação Especial.** Valente, J. A. org., NIED - UNICAMP Campinas, 1991.

UM MODELO PARA A IMPLEMENTAÇÃO DE FEDERAÇÕES DE TRADERS

A MODEL FOR THE IMPLEMENTATION OF TRADER FEDERATIONS

Prof. Dr. Edmundo Roberto Mauro MADEIRA*
Luiz Augusto de Paula LIMA JÚNIOR**

ABSTRACT

This article describes a proposal of a **TRADER** which is the computational object that receives service offers from other objects (called "servers") storing them in a data base. When another object (a "client") needs a computational service, it cam ask the trader about the service, and then the trader may return the "adress" of a server which offers that service. The grouping of traders to form Federations is also discussed presenting the advantages and a model for building such organization. Algorithms and typical scenaries to negotiate the federation contract are also introduced. Finally, the details of the implementation of the prototype of the trader are discussed. This prototype is in process of development and it is a part of the *MULTIWARE Platform* which aimes to provide an environment to support open distributed processing. The main algorithms and data structures used are also commented.

KEY WORDS: Trader, distributed system, Trader Federations,open distributed system.

RESUMO

Este artigo descreve uma proposta de um modelo para o **TRADER** que é o objeto computacional que recebe ofertas de serviços de outros objetos (chamados "servidores") colocando-as numa base de dados. Assim que um outro objeto qualquer (um "cliente") necessitar de um serviço computacional ele pode requisitar ao trader que então o informa a respeito da "localização" de algum servidor que oferece esse serviço. A união de traders em Federações também é discutida apresentando-se vantagens e modelos para a construção de tal organização, bem como algoritmos e cenários típicos para a negociação de contratos. Por fim, são apresentados com mais detalhes os esquemas de implementação do trader que está sendo implementado na UNICAMP como parte da *Plataforma MULTIWARE* que tem por objetivo criar um ambiente adequado para processamento distribuído aberto. Os principais algoritmos e estruturas de dados utilizados são comentados.

PALAVRAS-CHAVE: Trader, sistema distribuido, Federações de Traders,sistema distribuido aberto.

1. INTRODUÇÃO

O futuro da computação inclui o desenvolvimento de sistemas distribuídos e abertos. Sistemas distribuídos *homogêneos* já têm sido desenvolvidos e estudados extensivamente porque permitem que usuários compartilhem objetos que podem ser recursos periféricos e computacionais, serviços e informações. Esses sistemas consistem de várias estações de trabalho individuais e recursos periféricos (por exemplo, discos, impressoras) conectados por uma rede local (LAN). Estes computadores têm hardware e software similares e estão sob o controle de um sistema operacional distribuído. Através da interconexão de vários sistemas distribuídos locais para formar um grande sistema distribuído homogêneo, os usuários de cada sistema local se tornam capazes de compartilhar recursos que não estão disponíveis nos seus sistemas locais.

A necessidade de utilizar uma variedade de objetos e o crescimento da diversibilidade de hardware e software

(*) Professor do DCC - IMECC - UNICAMP
(**) Mestrando - DCC - IMECC - UNICAMP

fizeram com que a *heterogeneidade* se tornasse algo corriqueiro. Se usuários individuais estão conectados a um ambiente heterogêneo o número de objetos que eles podem acessar é significativamente maior do que aqueles disponíveis em suas redes locais (por exemplo, banco de dados, sistemas de arquivos, serviços de diretório, supercomputadores e outros hardwares especializados). Assim, através da interconexão de sistemas distribuídos locais e de diferentes computadores, um grande sistema heterogêneo e distribuído de computadores (sistema aberto) pode ser construído. Este sistema aberto é gerenciado por um sistema operacional aberto.

Apesar das vantagens e do seu potencial, sistemas distribuídos abertos trazem consigo várias dificuldades:

1. Um usuário de um sistema distribuído perde a autonomia que possuía em ambientes "stand-alone", e a necessidade de mecanismos de segurança se torna um verdadeiro desafio;

2. A heterogeneidade acaba com o acesso direto a um objeto, e se usuários querem permitir com que outros usuários tenham acesso aos seus objetos, esses acessos devem ocorrer em condições determinadas em comum acordo;

3. Como o número de objetos que oferecem serviços normalmente é muito grande, deveria existir um meio de localizar determinados servidores que satisfaçam uma lista de propriedades desejadas.

Como podemos então resolver todos estes problemas? A ISO[1] propôs uma solução introduzindo uma nova classe de servidores de nomes e gerenciadores de objetos conhecidos como **TRADERS** (19) (20) dentro do Modelo de Referência para Processamento Distribuído Aberto (RM-ODP) (15) (16) (17) (18) (7) (8).

"*Trading*", em ODP, pode ser considerado como sendo uma forma melhorada de nomeação e "binding" (acoplamento) (5). No caso de trading, o nome de um serviço pode ser substituído por uma descrição das propriedades desejadas de um servidor.

Poderíamos comparar o trader a um serviço de "classificados" por telefone onde pessoas que estão oferecendo produtos (ou serviços) informam a uma central (o trader) sobre os dados pertinentes àquele tipo de serviço. Assim que a central receber um telefonema de alguém procurando por aquele produto (ou serviço) ela se encarrega de achar a melhor oferta (se é que há alguma) que mais se adequa aos requisitos do cliente que ligou. A central passa então as informações (inclusive o número de telefone) do fornecedor do produto (ou serviço).

Num exemplo mais realista, consideremos uma rede heterogênea de computadores como na figura 1.

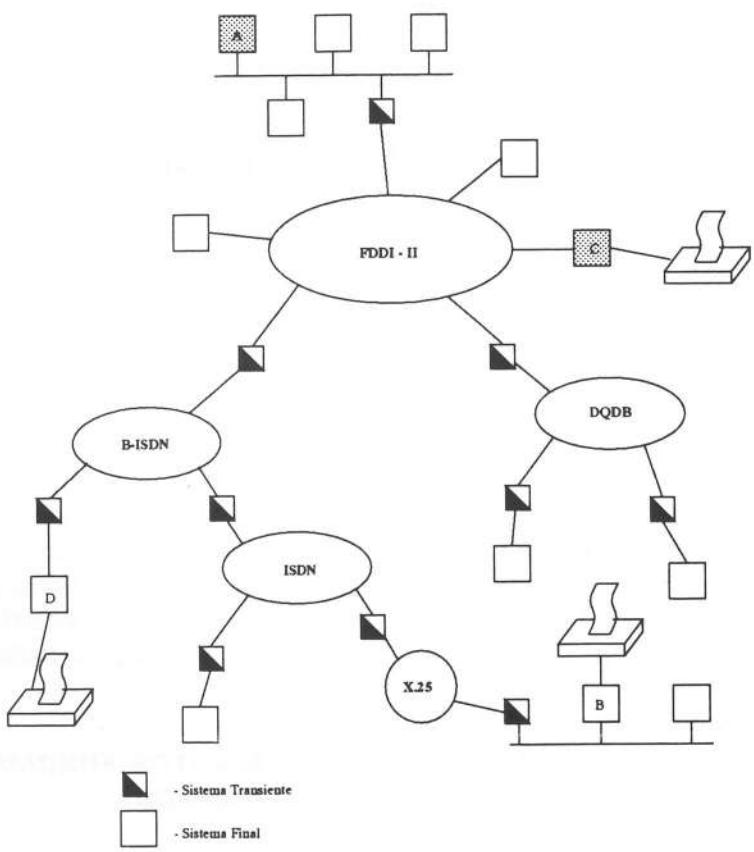


Figura 1 - Rede Heterogênea

(1) ISO: International Organization for Standardization

Se o usuário (objeto computacional ou ser humano) em A deseja utilizar uma impressora laser que esteja no "Centro de Computação" da empresa em que ele trabalha, cujo preço por página impressa seja mínimo, como descobrir o "endereço" de alguém que ofereça este serviço (se é que existe alguém e se é que A tem autorização para isso)? E ainda mais: como fazer isso de forma transparente, ou seja, de modo a esconder os detalhes da topologia da rede? No modelo ODP, é somente através da utilização do trader que objetos podem saber a respeito da existência de serviços os quais ele deseja requisitar. No exemplo, os servidores de impressão B, C e D podem ter informado previamente ao trader a respeito das características dos serviços de impressão que eles oferecem ("PRINTER" na figura 2). Assim, quando A faz o pedido de serviço ao trader, ele se encarrega de achar o servidor (digamos, C) que melhor se adequa às exigências feitas por A, acoplando A a C. O serviço exportado por C também está apresentado na figura 2.

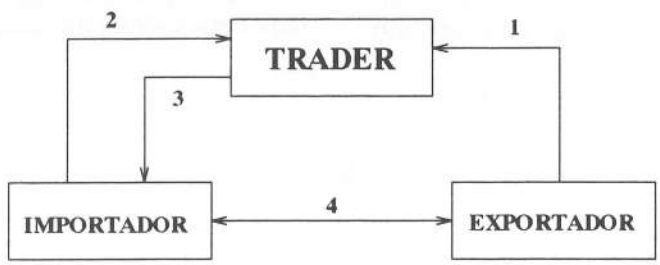
Em suma, a interação do trader com outros objetos (cliente e fornecedor) está representada na figura 3.

2. FEDERAÇÃO DE TRADERS - VISÃO GERAL

Uma Federação de Traders é uma coleção de traders cooperativos mas autônomos.

Tipo de Serviço (PRINTER)	Serviço Exportado por C
Printer type : DotMatrix, Laser	*Printer type* = Laser
Identifier : String	*Identifier* = "HP"
Localization : String	*Localization* = "Centro Comp. - Empresa de A"
Paper size : A4, A3 (default A4)	*Paper size* = A4
Cost per page : Float	*Cost per page* = 0.5

Figura 2 - Tipo de serviço e serviço exportado por C.



1. Exportador informa trader sobre serviços oferecidos.
2. Importador solicita serviços.
3. Trader informa importador a respeito de exportador do serviço solicitado.
4. Interação entre importador e exportador.

Figura 3 - Interações do trader com seus usuários

Através da federação, uma oferta de um trader componente poderá ser conhecida por uma audiência mais ampla e um trader componente da federação terá um mercado maior para suprir as suas necessidades.

Uma federação de traders também pode ser formada quando o número de objetos administrados por um trader se torna tão grande que não é possível que a administração sirva eficientemente aos seus usuários. Através do particionamento das responsabilidades administrativas do banco de dados do trader em traders cooperativos, uma federação de traders pode ser formada.

Numa federação, devem existir mecanismos para permitir cooperação entre traders que são autônomos, heterogêneos e distribuídos. Além do mais, os objetivos do ODP exigem a transparência de distribuição de modo que um usuário de um trader federado será associado a somente um trader e acessará transparentemente outros traders através daquele trader.

2.1. ASPECTOS FUNDAMENTAIS DE TRADERS FEDERADOS

Existem cinco princípios básicos que devem caracterizar o relacionamento entre os traders componentes de uma federação:

1. um componente não pode ser forçado a executar uma atividade para outro componente;

2. um componente deve ser livre para entrar e sair de uma federação;

3. um componente deve ser capaz de determinar quais os dados que ele deseja compartilhar com os outros;

4. um componente determina como ele vê e combina os dados existentes;

5. usuários locais, atividades e dados devem sofrer alterações mínimas (se é que há alguma) quando se incluem os mecanismos de federação.

Esses princípios devem reger toda formulação de modelos e algoritmos para a federação de traders.

2.2. UMA PROPOSTA DE MODELO PARA FEDERAÇÃO DE TRADERS

O modelo proposto para a federação de traders é o modelo descentralizado que tem a característica de garantir a **autonomia** de cada trader (que é um dos princípios básicos que regem a formação de federações) (3).

Para ser parte de uma Federação, um trader deve importar serviços de pelo menos um outro trader ou deve exportar serviços para pelo menos um outro trader na federação.

Um trader que importa serviços de um trader remoto tem um *contrato de importação* com aquele trader remoto. O contrato de importação define:

- os tipos de interfaces disponíveis no trader remoto;

- as regras para mapear requisições e respostas entre o trader local e o remoto de forma que sejam inteligíveis para ambos;

Um trader que exporta serviços para um trader tem um *contrato de exportação* com o trader remoto. O contrato de exportação define:

- a extensão do acesso ao banco de dados do trader local ao trader remoto;

- os tipos de interface disponíveis no trader remoto;

- as regras para mapear requisições e respostas entre o trader local e remoto de forma que sejam inteligíveis para ambos;

Para cada contrato de exportação existe um contrato de importação correspondente no trader remoto.

Um trader que exporta para um trader remoto oferece uma interface de "*Estabelecimento-Federação*"" e uma interface de "*Operações-Federadas*" para o seu trader remoto (19).

O procedimento para fazer uma requisição de serviço é o seguinte: um cliente de um trader procura por um serviço no banco de dados do trader e, se necessário, o trader local busca através dos seus contratos de importação. Se o tipo de serviço solicitado está disponível num trader remoto, então o trader local mapeia a requisição em uma requisição remota e envia para o trader remoto através da interface "Operações-Federadas" desse trader remoto. O trader remoto inicia uma busca no seu banco de dados, de acordo com o contrato de exportação correspondente. Quando a busca for completada, os resultados (transformados, se necessário) são retornados ao trader que está importando os serviços e por fim ao cliente que requisitou os serviços. A figura 4 mostra as interfaces entre traders via contratos.

3. ASPECTOS DE MODELAGEM E IMPLEMENTAÇÃO

3.1. MODELO

Em nosso trabalho identificaram-se basicamente dois grupos de atividades desempenhadas por um trader: o gerenciamento de bases relacionadas às informações (estáticas e dinâmicas) mantidas pelo trader e a execução propriamente dita das operações utilizando-se das informações armazenadas (figura 5).

O modelo que propomos consiste de um refinamento desses dois módulos considerando separadamente funções locais e aquelas relativas ao estabelecimento e "manutenção" de federações bem como separando em módulos diferentes o gerenciamento/obtenção das informações estáticas e dinâmicas. Assim chegamos a um modelo (figura 6) que supre toda a funcionalidade requerida de um trader (inclusive para federação).

O administrador local e o de federação (objetos ou humanos), que podem ser um só, são responsáveis pela definição e cumprimento de certas políticas de negociação locais e no nível de federação, respectivamente. O *administrador local* acrescenta novos tipos de serviços a um dado contexto entre outras atividades, por isso deve estar acoplado diretamente ao repositório de tipos. Já o *administrador de federação* é responsável por preparar o catálogo, requisitar o catálogo, decidir quan-

do e com quem federar-se, e aceitar, recusar ou formular propostas de contratos definindo assim políticas para o estabelecimento da federação.

Dois módulos são responsáveis pelo armazenamento e/ou obtenção de informações no trader. São eles:

MÓDULO DE INFORMAÇÕES ESTÁTICAS

Concentra as operações que manipulam as informações estáticas que correspondem a tipos de serviços (no Repositório de Tipos) e ofertas de serviços (no Diretório).

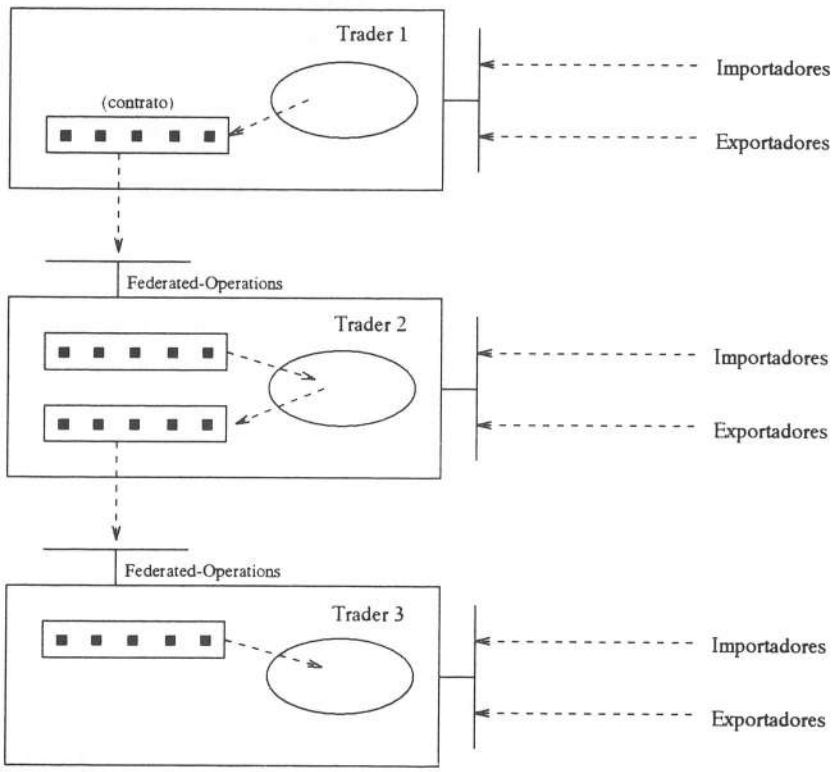


Figura 4 - Federação de traders

Figura 5 - Atividades básicas de um trader

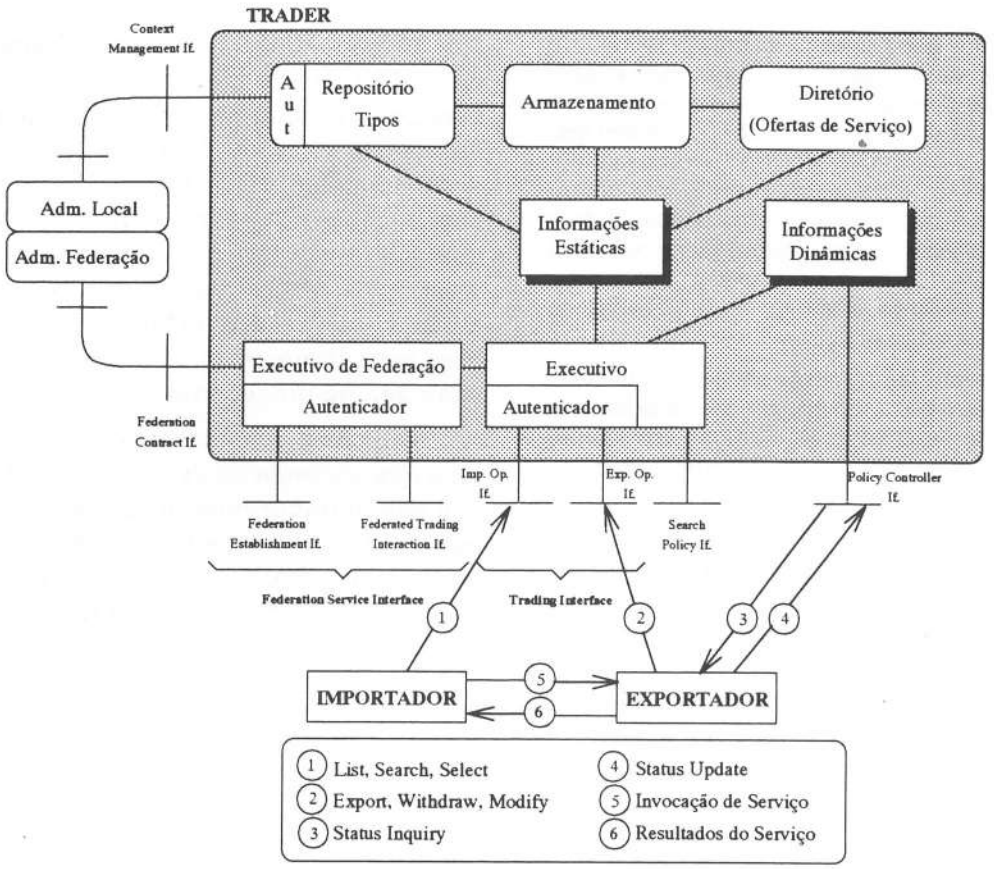


Figura 6 - Esquema interno de um trader

MÓDULO DE INFORMAÇÕES DINÂMICAS

Responsável por atualizar as propriedades dinâmicas de uma dada oferta de serviço. Esse módulo entra em contato com o *controlador de política* do objeto exportador para obter as informações dinâmicas. Um exportador de um serviço não tem necessariamente que ter o controlador de política associado à sua oferta de serviço. Se o identificador do controlador de política é nulo (indicando sua inexistência) nenhuma operação é feita. Pode ser implementado usando o X.700 da ITU-T.

O *Repositório de Tipos* suporta o armazenamento (utilizando-se do módulo de Armazenamento) e a combinação dinâmica de tipos de serviços para indicar, por exemplo, quando dois tipos são equivalentes, quando um tipo é subtipo de outro e assim por diante.

O *Diretório* armazena as ofertas exportadas para o trader. Pode ser implementado utilizando-se o serviço de Diretório X.500 da ITU-T.

O módulo *Executivo* é o responsável por executar as operações locais do trader utilizando-se de informações estáticas e dinâmicas. E o "*Executivo de Federação*" executa as operações do trader via federação e também permite o estabelecimento da federação através dos contratos.

O *Autenticador* é o módulo que verifica a permissão do importador/exportador para executar a dada operação. Também é responsável por aplicar as restrições de acesso às bases de dados do trader, como definido pelo(s) administrador(es).

3.2. IMPLEMENTAÇÃO

Idealmente a implementação do modelo descrito na seção 3.1. deveria ser feita sobre a plataforma ODP utilizando-se das estruturas, transparências e facilidades para a construção de aplicações distribuidas lá fornecidas. Uma outra alternativa seria o uso de plataformas como ANSAware[2] (23) (24) (25), DCE[3] (9) (10) (11) (12) (13) (14) ou ORB[4] (21) (22).

(2) ANSA: Advanced Network Systems Architecture
(3) DCE: Distributed Computing Environment
(4) ORB: Object Request Broker

Inicialmente foi construído um protótipo de um trader diretamente em cima do sistema operacional UNIX. São utilizadas as chamadas de sistemas padrão e chamadas a procedimentos remotos (RPC's). Além disso, utiliza-se na implementação "*sockets*" e uma simulação de "processos leves" ("threads"). Com isso, procura-se tornar o protótipo o mais portável possível para outros ambientes dos quais a corrente implementação possa vir a fazer parte no futuro. O próximo passo será transportar o modelo implementado para utilizar as facilidades de uma das plataformas citadas acima.

Estão implementadas somente as funções básicas de um trader que permitem com que uma federação seja formada. A implementação dos algoritmos para a federação ainda se encontra em fase de desenvolvimento.

3.2.1. COMUNICAÇÃO - O "BROKER"

Esquematicamente o protótipo construído está representado na figura 7.

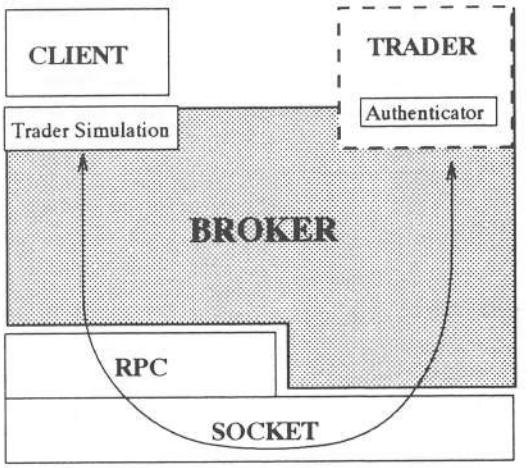


Figura 7 - Ambiente construído juntamente com o trader

O módulo de *simulação do trader* faz com que o cliente do trader (um importador ou exportador de serviços) tenha a impressão de que todas as suas operações estão sendo feitas localmente, tornando assim totalmente transparente todos os detalhes da comunicação. Chamamos a parte da implementação que lida com a transparência para a comunicação via rede de "Broker", um termo Já consagrado pela CORBA[5] (4). O "Broker" tem a função de transformar uma chamada a uma função local em uma chamada de procedimento remoto.

Um cliente é construído utilizando uma biblioteca que esconde os detalhes de toda a comunicação com o trader remoto. Cada invocação de uma operação do trader é feita localmente ao "*Broker*" do lado do cliente, que, por sua vez, comunica-se com o "*Broker*" do lado do trader via RPC passando os seguintes parâmetros:

- identificador da interface;
- nome de quem está requisitando a função (cliente);
- lista de parâmetros adicionais (pode ser vazia).

que são fixos para todas as operações. Caso os parâmetros a serem transmitidos para o trader sejam muito complexos, então o "*Broker*" se encarrega de transmiti-los via rede através de "*stream sockls*". Se não houver permissão para dado cliente efetuar a dita operação, os parâmetros complexos nem são enviados, o que melhora o desempenho nestes casos.

Está também embutido no "*Broker*" uma implementação básica do mecanismo de interfaces. Assim, ao chamar uma determinada função, é imprescindível que se especifique em qual interface aquela função está localizada. O "*Broker*" do lado do trader, antes de tudo, sempre verifica a existência de tal operação na interface especificada, e então chama localmente a função do trader correspondente.

O "*Broker*" é também responsável por retornar os resultados das operações. Aqui novamente, se os resultados forem complexos (no caso da operação "search", por exemplo), eles são transmitidos para o cliente através de "*stream sockets*".

3.2.2 O TRADER

O protótipo está ainda em fase de desenvolvimento para incorporar operações para o gerencia-

(5) Um "broker" é uma pessoa que faz um determinado serviço no lugar de outra pessoa. (CORBA: Common Object Request Broker Architecture)

mento de federação de trader, porém, no estágio atual, estão oferecidas as operações de "*export*", "*withdraw*", "*search*" e "*list_offer__details*", que são essenciais.

Antes que qualquer operação seja executada, um módulo chamado "autenticador" verifica a autorização do cliente em executar a determinada operação. Se não há permissão, um código de erro apropriado é retornado, caso contrário, a operação é executada.

3.3. FEDERAÇÃO DE TRADERS

A comunicação entre dois traders para se construir uma federação ocorre da mesma forma descrita para a comunicação entre um trader e o seu cliente[6].

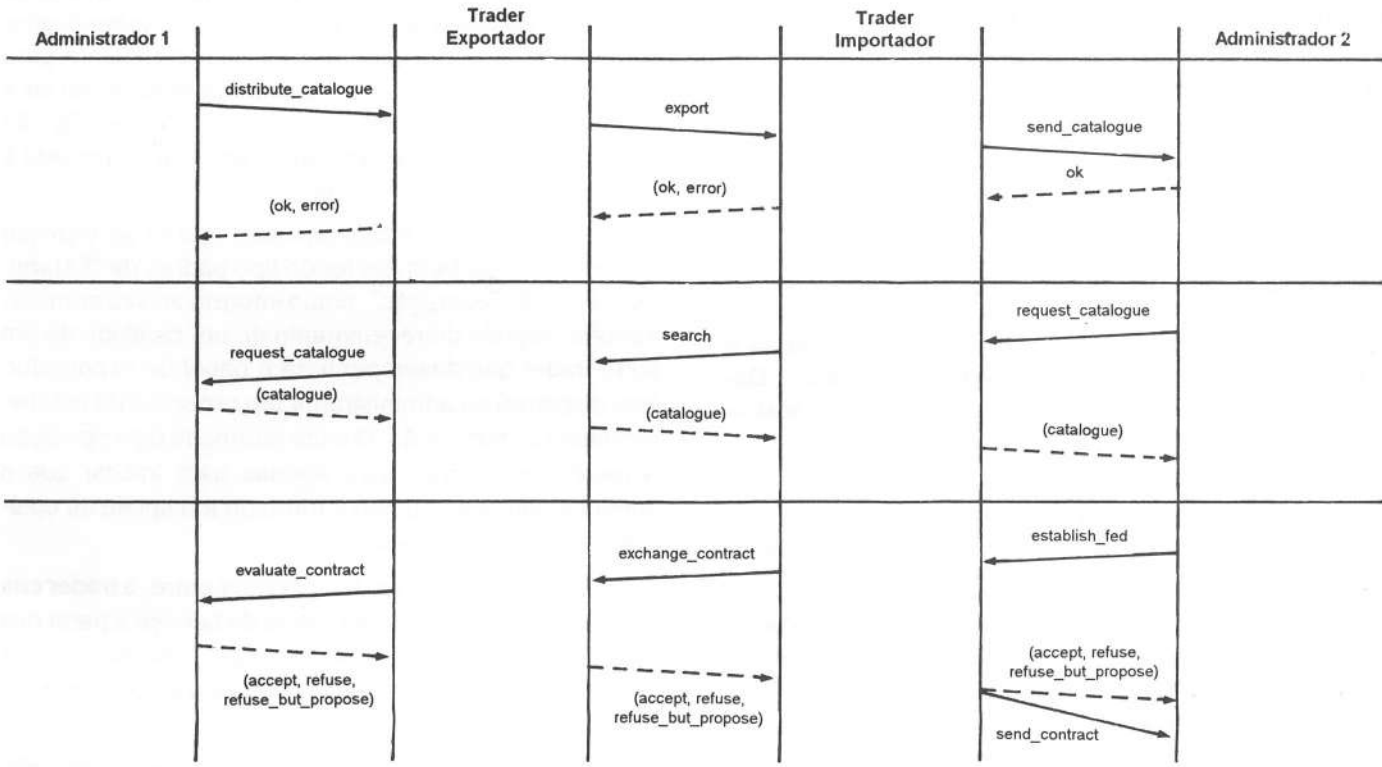


Figura 8 - Cenário para a negociação e o estabelecimento do contrato

Para o estabelecimento do contrato, é necessária a intervenção do *Administrador de Federação* (veja figura 6), pois é dele que partem as decisões de formar a federação e de avaliar e propor contratos.

O esquema da figura 8 apresenta o cenário para a negociação e o estabelecimento de um contrato de federação entre dois traders, onde um faz o papel de "exportador" de serviços e o outro de "importador" de serviços.

Antes de mais nada, para se estabelecer um contrato de federação entre dois traders é preciso que o trader que fará o papel de importador obtenha o *catálogo* do trader exportador. E isto pode ocorrer de duas maneiras.

No primeiro caso, o *administrador 1* (que é o administrador da federação do trader exportador na figura 8) decide enviar o seu catálogo para um potencial trader importador. O trader exportador ao receber a requisição para distribuir o catálogo, juntamente com o próprio catálogo, age como um cliente qualquer do trader importador usando a operação "*export*" tendo como propriedades do serviço o catálogo, e como tipo de serviço o tipo padronizado "ESTABELECIMENTO DE FEDERAÇÃO". O que é retornado ao *trader importador*, *ao trader exportador e ao administrador 1* é apenas um indicador de que o administrador do trader importador está ciente ou não do catálogo enviado.

Uma outra maneira de fazer com que o catálogo seja conhecido pelo administrador do trader importador, é quando o próprio administrador faz um pedido de catálogo a um potencial trader exportador. Essa opera-

(6) No processo de negociação do contrato de federação, somente se torna necessária a inclusão de uma nova operação ("Exchange Contracts"). Toda a troca de informações que precede a "assinatura" do contrato é feita utilizando-se as operações normais do trader (alterando-se os algoritmos).

ção, no trader exportador, é transformada num "*search*"" com o tipo de serviço padronizado "ESTABELECIMENTO DE FEDERAÇÃO". O catálogo é então passado do *administrador 1* até o *administrador 2* (veja figura 8)

Uma vez que o administrador do trader importador possui o catálogo, ele pode agora iniciar a negociação do contrato de federação fazendo uma proposta ao administrador do trader exportador. Este pode aceitá-la plenamente, recusá-la ou recusá-la e ainda fazer uma contraproposta com um contrato reduzido. Deste modo um contrato de federação pode ser estabelecido e então as operações federadas podem ser executadas.

3.3.2. OPERAÇÕES FEDERADAS

O módulo "Executivo de Federação" (veja figura 6) do trader exportador, ao receber a requisição de qualquer operação, interpreta o seu contrato de exportação correspondente aplicando as restrições ali contidas e passa a execução propriamente dita ao módulo "Executivo". Depois disto, ele se encarrega de retornar os valores (transformados, se necessário) ao trader importador.

No lado do trader importador, o módulo "Executivo de Federação" é acionado pelo módulo "Executivo" quando este identificar a necessidade de alguma operação federada.

De um modo geral, o papel do "Executivo de Federação" do trader importador é transformar uma requisição local em uma operação federada que é passada ao trader exportador. As operações executadas via federação, passam pelo módulo "*Executivo de Federação*" que se encarrega de interpretar o contrato de exportação aplicando as restrições e operações combinadas.

Os algoritmos das operações que formam o conjunto mínimo a ser fornecido aos clientes de um trader (importador, exportador ou outro trader) estão comentados a seguir.

OPERAÇÕES DISPONÍVEIS NA INTERFACE "EXPORTER OPERATIONS"

As operações disponíveis nesta interface são:

EXPORT: corresponde à criação de uma oferta de serviço no diretório do trader;

WITHDRAW: remove uma oferta de serviço do diretório;

MODIFY: corresponde à uma operação de "withdraw" seguida de uma operação de "export" feitas atomicamente

(7) Um contexto pode ser considerado como sendo um "subdiretório".

As operações "*withdraw*" e "*modify*" não sofrem modificações para incorporar os mecanismos de federação propostos neste trabalho, por isso não discutiremos aqui o seu algoritmo.

A OPERAÇÃO "EXPORT"

Antes de mais nada, o trader que recebeu uma solicitação da operação de "*export*" aciona o autenticador que verifica a permissão do determinado cliente para executar essa operação. Se não há permissão ou se a interface através da qual o cliente fez a requisição não possuir a operação "*export*", então um erro apropriado é retornado e nada mais é feito.

Depois disso, o trader analisa o tipo de serviço que está sendo exportado. Se for do tipo padrão de "*Estabelecimento de Federação*", então informa ao seu administrador a respeito do recebimento de um catálogo de um outro trader que desempenhará o papel de exportador. Isso disparará no administrador um processo de estabelecimento da federação. O valor retornado pela operação "*export*", neste caso, será apenas para indicar que o administrador está ou não informado a respeito do catálogo do trader exportador.

Se o tipo de serviço for qualquer outro, o trader cria em sua base de dados uma oferta de serviço a partir dos parâmetros recebidos do cliente, retornando um identificador da oferta de serviço que é o único num dado contexto[7].

Um código de erro será retornado se a operação não puder ser executada por algum motivo.

OPERAÇÕES DISPONÍVEIS NA INTERFACE "IMPORTER OPERATIONS"

As operações disponíveis nesta interface são:

LIST_OFFER_DETAILS: retorna os detalhes de uma determinada oferta de serviço;

SEARCH: retorna todas as ofertas de serviços que satisfazem os requisitos do importador;

SELECT: seleciona a melhor oferta de serviço (a partir de critérios determinados) dentre as que satisfazem às propriedades requeridas pelo importador;

Aqui novamente a operação "*search*" é a única alterada para permitir o estabelecimento de federação e as operações federadas.

A OPERAÇÃO "SEARCH"

Após verificada a permissão do cliente para efetuar a operação, o tipo de serviço pedido é analisado. Se for do tipo padrão de "*Estabelecimento de Federação*", o trader pede ao seu administrador que crie um catálogo com as possíveis operações via federação e restrições de acesso. Então retorna o catálogo ao cliente (que é um outro trader, neste caso) juntamente com o identificador para a sua interface de estabelecimento de federação, se a operação for bem sucedida.

Se o tipo de serviço pedido não for de estabelecimento de federação, o trader faz a busca dentro de um escopo adequado para obter as ofertas de serviço que satisfazem aos critérios pedidos pelo cliente. É nesse momento que são obtidas tanto as informações estáticas quanto as dinâmicas das ofertas de serviços.

Se não acha ofertas de serviço adequadas localmente, verifica os seus contratos para fazer uma requisição de busca ao "*Executivo de Federação*" que se encarregará de executar o mesmo "search" no trader remoto.

É retornada ao cliente uma lista de ofertas de serviços com as propriedades pedidas juntamente com os identificadores das interfaces onde os serviços estão disponíveis.

A OPERAÇÃO "EXCHANGE-CONTRACT"

Esta operação está disponível na "*interface de estabelecimento de federação*". Através desta operação, um trader importador poderá fazer uma proposta de um contrato para a federação. O trader exportador ao receber esta proposta, pede ao seu administrador para avaliá-la após verificar permissões e interfaces. Se a proposta de contrato for aceita pelo administrador, o trader cria uma nova interface para interações federadas e exporta para a sua própria base de dados o tipo de serviço padronizado "INTERAÇÃO DE NEGOCIAÇÕES FEDERADAS" tendo o contrato de exportação como sendo a propriedade do serviço. O identificador da interface é então retornado ao trader importador se não houver problemas. Caso contrário, um código de erro é retornado.

O protótipo implementado a partir destes algoritmos segue o modelo proposto na figura 6 e está incluido no projeto da plataforma Multiware(1).

4. CONCLUSÃO

O **Trader** desempenha um papel muito importante em ambientes distribuídos e abertos, pois em um Sistema Distribuído Aberto é altamente desejável que existam meios de fazer a seleção dinâmica (em tempo de execução) de serviços computacionais que satisfaçam a determinadas propriedades. É através do trader que importadores de serviços (clientes) podem entrar em contato com exportadores de serviços (servidores) no modelo ODP.

O presente trabalho propôs um modelo de implementação do trader dando atenção especial aos aspectos que dizem respeito à **"Federação de Traders"** apresentando-se algoritmos e detalhes de implementação do protótipo construído.

A maior dificuldade encontrada na fase de implementação foi com relação à parte de comunicação entre objetos através da rede. Foi necessária a construção de uma camada (o "*BROKER*", figura 7) que toma os aspectos de comunicação transparentes para o trader e especialmente para o cliente do trader. Com o futuro transporte do protótipo do trader para o ambiente *Multiware* (1), a camada "BROKER" (aqui, de uso específico do trader) será substituida por outra camada que fornecerá todas as facilidades de comunicação necessárias para atender não só ao trader, mas também a um objeto computacional genérico. Para tanto, o "BROKER" do trader será o resultado da compilação do conjunto de interfaces do trader.

Também, a inexistência de "binding" dinâmico no ambiente em que nos propusemos a desenvolver o protótipo, impôs algumas restrições na implementação do modelo criado. Este problema foi parcialmente contornado com a introdução do código que simula o conceito de interfaces e que faz parte do "BROKER".

AGRADECIMENTOS

Agradecemos à CAPES e à FAPESP pelo apoio financeiro durante o desenvolvimento do projeto.

REFERÊNCIAS

(1) *Madeira, E. R. M.; Mendes, M.* "Plataforma Multiware: Projeto e Desenvolvimento da Camada Multiware", XII SBRC - Curitiba, maio, 1994.

(2) *Mendes, M. J.; Loyola, W. P. D. C.; Madeira, E. R. M.* "Demos:A Distributed Decision-Making Open Support System" -4th IEEE Workshop on Future Trends in Distributed Computing Systems, Lisboa, Portugal, pp 208-214, Sept,1994.

(3) *Bearman, M.* "ODP Trader" - Proceedings of the ICODP, pp 19-23, Sept, 1993.

(4) *Vogt, F.; Andrae, C.* "Middleware for Distributed Applications Support : ODP and/or CORBA, Proceedings of the ICODP, pp 405-430, Sept., 1993.

(5) "Broadening the User Environment with Implicit Trading" Proceedings of the ICODP, pp 129-140, Sept.,1993.

(6) *Meyer, B.; Popien, C.*;"Object Configuration by ODP Traders" Proceedings of the ICODP, pp 425-430, Sept.,1993.

(7) *Tschammer, V.*, "CODE - Cooperating Open System, Open Distributed Processing, Distributed Computing Environment Experiments and Projects", Nov.,1992.

(8) Tschammer, V.; Mendes, M. J.; Souza, W. L.; Madeira, E. R. M.; Loyolla, W.,"Processamento Distribuído Aberto e o Modelo RM-ODP/ISO", XI SBRC,1993.

(9) "OSF Distributed Computing Environment" - Sept 1990.

(10) Fauth, Dr. Dietmar; Gossels, Jonathan; at all, "Distributed Computing Environment Evaluation Team-OSF", May, 1990.

(11) "Distributed Computing Environment Overview" - OSF.

(12) "Directory Services for a Distributed Computing Environment", OSF, Sept.,1990.

(13) "File Systems in a Distributed Computing Environment", OSF, Sept.,1990.

(14) "Remote Procedure Call in a Distributed Computing Environment",OSF, Oct.,1990.

(15) "ISO/IEC JTC 1/SC 21/N 7053" Basic Reference Model of ODP - Part 1: Overview and Guide to Use.

(16) "ISO/IEC DIS 10746-2 - ITU-T Draft Rec. X. 902 - Feb. 1994" Basic Reference Model of ODP - Part 2: Descriptive Model.

(17) "ISO/IEC JTC 1/SC 21/N 1076-3.2 - ITU-T Draft Rec. X. 903 - Feb.,1994" Basic Reference Model of ODP - Part 3: Prescriptive Model.

(18) "ISO/IEC 21/N 7056" (Recomendation X. 905) Basic Reference Model of ODP - Part 5: Architectural Semantics.

(19) "ISO/IEC JTC 1/SC 21/N 7047, 1992-06-30" Working Document on Topic 9.1 - ODP Trader

(20) "ISO/IEC JTC 1/SC 21 - Nov. 1993" Information Technology - ODP Trading Function

(21) HP Company, Sun Microsystems, inc, "The Object Management Group, Object Rquest Brocker, RFP joint response",1991.

(22) HP Company, Sun Microsystems, inc, "Distributed Object Management Facility Core Especification",1991.

(23) "ANSA: An Engineer's Introduction to the Architecture" - Release TR.03.02-Architecture Projects Management Limited Nov., 1991.

(24) "ANSA Work-Programme - Phase III" - Issue I - Architecture Projects Management Limited, June, 1991.

(25) "The Open Systems Newsletter - ANSA: Objects Pioneers" - Technology Appraisals Ltd, V. 7, Issue 2, Feb., 1993..

GERAÇÃO DE PROGRAMAS PARA ROBÔS INDUSTRIAIS EM AMBIENTE CAD

INDUSTRIAL ROBOT PROGRAM GENERATION VIA CAD SYSTEM

Carlos Norberto VETORAZZI JÚNIOR*
Prof. Dr. Geraldo Nonato TELLES**

ABSTRACT

The work described there is the development of a system to automatically generate robot programs for PCB (Printed Circuit Board) components insertion tasks, with the input of CAD files describing the lay-out of the robotic cell, and some files with information about the components of assembly task. The system plays the hole of finding the cordinates to be programmed - wich takes a lot of improductive time if manually done - and check for interference in the robotic insertion. This can lead to a more flexible way to programming a robot.

KEY WORDS: Robots, programs, automation, CAD - Computer Aided Design.

RESUMO

É descrito o desenvolvimento de sistema que gera automaticamente programas para inserção de componentes eletrônicos em placas de circuito impresso, a partir de arquivo CAD que descreve o "lay-out" da célula de fabricação, além de arquivos com informações sobre os componentes e as tarefas de montagem. O sistema determina as coordenadas a serem programadas - o que demandaria um tempo improdutivo grande se feito manualmente - e verifica a ocorrência de interferência na tarefa de inserção. O sistema desenvolvido possibilita uma flexibilização na programação de robô.

PALAVRAS-CHAVE: Robôs, programas, automação, CAD - Computador Auxiliando o Desenho.

INTRODUÇÃO

A montagem manual de componentes em cartões de circuito impresso (PCBs) é extremamente tediosa, sendo que atualmente utiliza-se com vantagens robôs industriais para essas tarefas (du Feu 1986),(Supinski 1991). Os robôs empregados nessas atividades são normalmente previamente programados através de linguagens textuais, e nestes programas encontram-se os valores das coordenadas, dadas no espaço de trabalho cartesiano do robô.

O sistema aqui apresentado utiliza como plataforma um microcomputador padrão PC, utilizando o sistema CAD AutoCAD R10 da Autodesk como fonte de dados. Os testes de validação foram feitos em um robô IBM 7535 do tipo SCARA (***), com linguagem AML para programação do robô (****).

A motivação para o desenvolvimento desse sistema reside na dificuldade de se determinarem corretamente as coordenadas que deverão constar no programa em AML. Essa obtenção é feita normalmente utilizando-se o robô como digitalizador dessas coordenadas, o que além de levar um tempo considerável, utiliza o robô de maneira improdutiva. Mas essa determinação poderia ser feita a partir de informações previamente disponíveis em arquivos CAD. Esses arquivos seriam os desenhos dos componentes, do alimentador, do dispositivo de fixação, das placas e do "lay-out" da célula, onde estariam as relações entre todos os elementos envolvidos. Se esses desenhos estão em uma escala adequada, e um desenho CAD não é mais do que um plano cartesiano onde cada

(*)Carlos Norberto Vetorrazi Jr. - Doutorando em Eng. Mecânica - UNICAMP.
(**) Geraldo Nonato Telles - Professor convidado do programa de Mestrado no convênio PUCCAMP/UNICAMP.
(***) SCARA: Selective Compliance ARticulated Arm
(****) AML: A Manufacturing Language

elemento do desenho pode ser descrito em termos de coordenadas desse plano cartesiano, então podemos "re-irar" as coordenadas para o programa desses desenhos.

Nesse projeto existe uma simplificação que consis-e no enfoque bidimensional (2D) do problema, tendo em vista que o robô em questão não tem servo-controle sobre o eixo Z; os parâmetros para este eixo podem ser apenas alto ("UP") e baixo ("DOWN"), tendo-se que obrigatoriamente projetar os alimentadores e dispositi-vos de fixação das placas de maneira que os componen-es são carregados e montados em uma mesma cota Z (mecanicamente determinada).

Esse projeto também não prevê o controle para se evitarem colisões, havendo necessidade de eventuais depurações (manuais) do programa após a simulação gráfica do mesmo, inserindo-se pontos intermediários (via-points), de forma a se evitar colisões da garra do robô com os outros dispositivos.

A utilização desse sistema é feita "off-line", indo de encontro a modernas tendências (inclusive Engenharia Simultânea), onde as atividades relativas à manufatura são associadas e concomitantes com as atividades de projeto, compartilhando uma mesma base de dados, havendo uma integração CAD/CAM, cujo objetivo final é o CIM(Jacobs 1991).

A automatização da montagem de componentes eletrônicos através de equipamentos dedicados (automação rígida) só é justificada para grandes volu-mes, tendo em vista as dificuldades de preparação dos sistemas ("set-up"). Assim, onde temos uma produção de menos de 100.000 PCBs por ano, ou mais de 100 tipos diferentes de PCBs(Supinski 1991), temos necessidade de flexibilidade, justificando o uso de sistemas robotizados programáveis, sendo importante a geração rápida e precisa de programas.

O emprego de Robôs não está livre de problemas, entretanto. Como vimos, existe a necessidade de uma integração CAD/CAM, de maneira a racionalizar a gera-ção dos programas para a manufatura.

OS ARQUIVOS CAD

É possível extrair as coordenadas de arquivos CAD. No sistema CAD utilizado é possível operar-se com "blocos" ("AutoCAD Reference Manual"). Cada bloco é um conjunto de entidades gráficas (retas, círculos), asso-ciadas como uma única entidade (o bloco). Assim, o desenho de um componente pode ser tratado como um bloco, bem como o desenho de uma placa. O comando ATTEXT (ATTribute EXTraction) do AutoCAD gera uma lista dos blocos utilizados em um desenho, com as respectivas coordenadas. Então, ao tratarmos os elementos (componentes, placas) como blocos em um desenho de "lay-out", podemos determinar as coordenadas de cada bloco, e estas coordenadas podem ser associadas com as coordenadas reais dos elementos na tarefa de montagem (Vetorazzi Jr., e Telles G.N. 1994).

Isto é feito construindo-se dois arquivos CAD (desenhos) de "lay-out" da montagem. Esse "lay-out" corresponde a uma representação do espaço de trabalho do robô. Então os desenhos dos elementos envolvidos no processo de montagem são posicionados (como blocos) nesse "lay-out", de maneira a representar a situação real. Um "lay-out" seria equivalente ao estado inicial do sistema, desenhos do alimentador e dos componentes. O outro "lay-out" seria equivalente ao estado final do sistema, quando os componentes encontram-se montados nas placas; então a montagem é feita posicionando-se o dispositivo de fixação no espaço de trabalho, as placas no dispositivo de fixação e os componentes nas placas (Vetorazzi Jr.,C. N. e Telles G. N. 1994) e (Vetorazzi Jr., C. N.1994)(figura 1).

Então utilizamos o comando ATTEXT para cada arquivo de "lay-out" e temos as coordenadas de cada bloco utilizado em cada "lay-out". As coordenadas que são importantes são as que correspondem aos componentes que não estão no alimentador e quando estão montados nas placas. Essas coordenadas correspondem àquelas que temos que colocar no programa do robô e que informam a este onde deve ir buscar os componentes e para onde deve levá-los. Em princípio, as outras coordenadas não interessam. Os desenhos do alimentador, do dispositivo de fixação e das placas, são usados como auxiliares para o correto posicionamento dos componentes, já que na situação real eles realmente estarão no alimentador e serão colocados nas placas, mas o que realmente interessa são as coordenadas dos componentes. A figura 2 mostra um arquivo típico gerado pelo comando ATTEXT, e a figura 3 mostra o desenho correspondente.

Essas coordenadas correspondem aos sistemas de referência utilizados para desenhar cada bloco em relação ao sistema de referência do "lay-out", que deve corresponder ao do próprio robô. Conseqüentemente, esses sistemas de referência devem ser criteriosamente escolhidos, sendo interessante que correspondam a algum atributo físico do elemento que será usado na montagem (Vetorazzi Jr.,C.N. 1994) (por exemplo um pino no componente e um furo no cartão) (figura 4). A seguir é necessário "corrigir" essas coordenadas em função da localização do sistema de referência, do tamanho do componente, do tamanho e da orientação da garra e da orientação do componente.

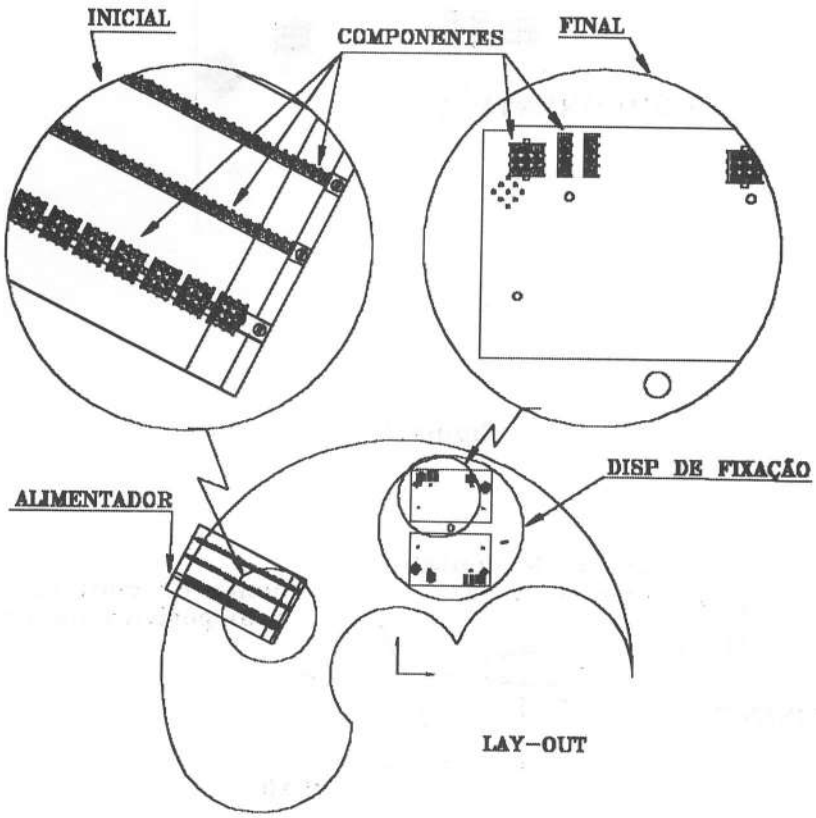


Figura 1

Nivel de aninham.	Nome do Bloco	Pos x	Pos y	Pos z	Num de ordem	Angulo r	Nome do componente
1,	'COMP1',	-1.96,	79.51,	0.00,	1,	0.00,	'componente1'
1,	'COMP1',	135.12,	76.86,	0.00,	2,	0.00,	'componente1'
1,	'COMP1',	175.65,	59.74,	0.00,	3,	315.00,	'componente1'
1,	'COMP1',	-15.96,	65.51,	0.00,	4,	315.00,	'componente1'
1,	'COMP2',	27.94,	79.51,	0.00,	5,	0.00,	'componente2'
1,	'COMP2',	44.34,	79.51,	0.00,	6,	0.00,	'componente2'

Figura 2

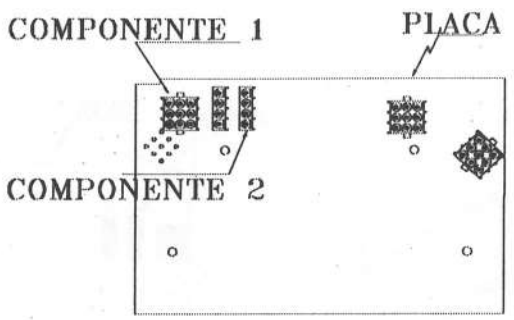


Figura 3

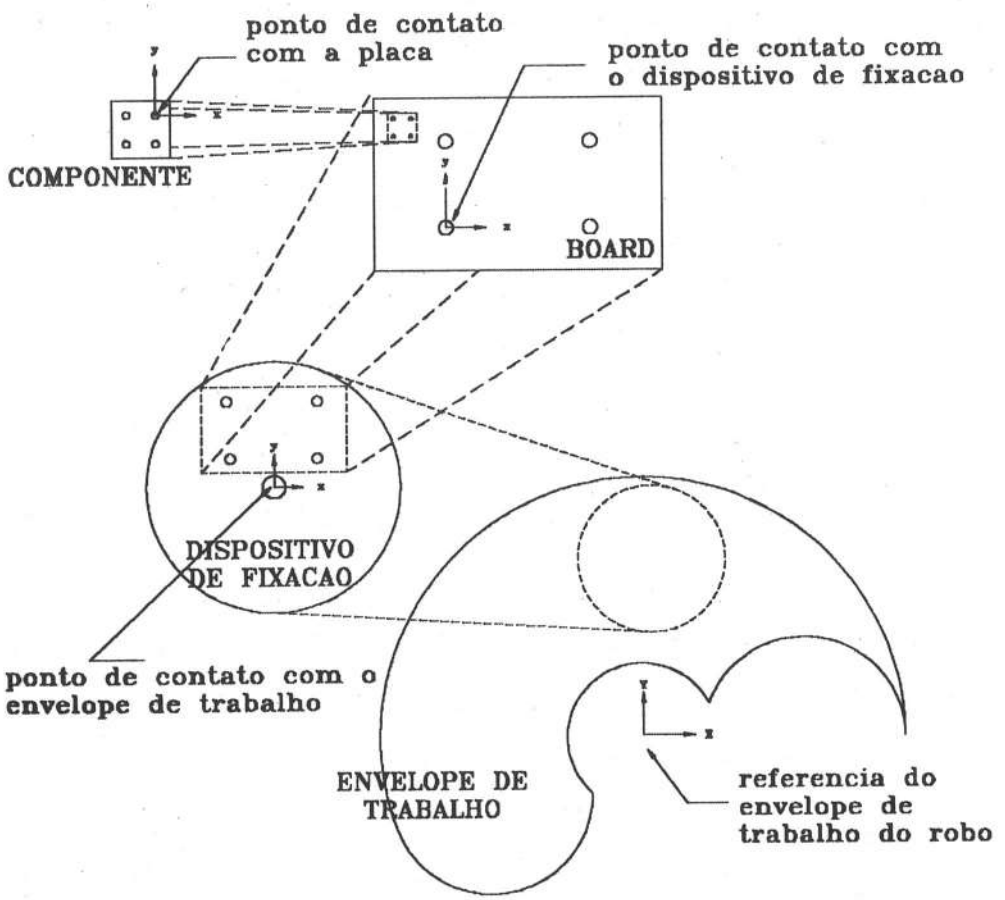


Figura 4

Então ajusta-se essas coordenadas de maneira que correspondam à coordenada da garra do robô. Com estas informações pode-se então determinar se vai haver interferência entre a garra e componentes previamente montados, o que implica na definição de algumas limitações e se necessário, na determinação da seqüência de inserção dos componentes, ou mesmo na indicação de componentes a serem inseridos manualmente. Atualmente essas informações são fornecidas manualmente pelo usuário, por ocasião do desenho dos elementos.

Outra atribuição do sistema (Vetorazzi Jr., C.N. 1994) é a definição de pontos intermediários, para a otimização do ciclo de montagem. Próximo a cada ponto-alvo, é definido um ponto intermediário, e entre estes, o robô se movimenta com alta acurácia (o que implica baixa velocidade), e entre os pontos intermediários o robô se movimenta com alta velocidade (o que implica baixa acurácia). Uma maior acurácia nas proximidades dos pontos-alvo reduz a possibilidade de erros do robô para atingir esses pontos, e uma maior velocidade entre os pontos intermediários faz com que o tempo de ciclo seja menor.

O SISTEMA

As coordenadas estão prontas para serem incorporadas ao programa. O programa é um conjunto de subrotinas específicas (uma subrotina para pegar o componente, outra para inserir o componente, etc). As subrotinas são as mesmas para várias montagens diferentes, somente as coordenadas mudam, então temos um problema do tipo "complete a sentença", onde os espaços em branco correspondem às coordenadas. E as sentenças com os espaços seriam comandos da linguagem, formando um "esqueleto" a ser preenchido.

Na verdade, este "esqueleto" do programa é interno ao sistema, e não um elemento a parte. Isto não é uma boa prática, porque é interessante escreverem-se pré-processadores para outras linguagens, de forma a se gerar um programa genérico (internamente ao sistema), que contenha a lógica do processo, e então seria possível gerar um programa em uma linguagem específica, traduzindo o programa genérico para esta linguagem, mediante um pré-processamento.

O sistema tem como base um menu para o direcionamento do trabalho, onde cada opção aciona um módulo específico. São três módulos principais; no primeiro escolhem-se os arquivos de "lay-out", e é feita uma preparação dos dados (agrupamento, ordenação); no segundo módulo são feitos todos os cálculos de coordenadas, verificação de interfaces, determinação de seqüência de inserção e determinação das coordenadas "corrigidas"; no terceiro módulo é feita a montagem do programa do robô, gerando os códigos do programa e incluindo as coordenadas.

As figuras 5 e 6 mostram dois programas gerados pelo sistema. A figura 5 corresponde a uma situação onde não há restrições devido a interferências, possibilitando o uso da primitiva "ITERATE" da linguagem AML/E, agrupando os elementos do mesmo tipo e tornando os programas mais concisos.

A figura 6 corresponde a uma situação onde há restrições.

```
- DEFINICOES DO ARQUIVO DEFAULT.AML :

PRECISAO : NEW 1;
VELOCIDADE : NEW 10;
ESPERA : NEW 1.5;
HOME : NEW PT( 650, 0, 0 );

-- COORDENADAS DOS ALIMENTADORES

S1  : NEW PT( -93.99, 358.23,-135.22); -- componente1
S2  : NEW PT( -125.31, 410.39,-135.22); -- componente2

-- JA EXISTE UM ALIMENENTADOR PARA O TIPO ABAIXO
-- S2  : NEW PT( -163.64, 448.43,-135.22); -- componente2

-- COORDENADAS DOS COMPONENTES NA PLACA

T1_3  : NEW PT(  23.29, 405.82, -45.00); -- componente1
T1_4  : NEW PT(  122.10, 500.88, -45.00); -- componente1
T1_5  : NEW PT(  169.86, 514.53,  0.00); -- componente1
T1_9  : NEW PT(  341.21, 325.18, 135.00); -- componente1
T1_10 : NEW PT(  242.40, 230.12, 135.00); -- componente1
T1_11 : NEW PT(  194.64, 216.47,-180.00); -- componente1
T2_7  : NEW PT(  56.03, 440.68, -45.00); -- componente2
T2_13 : NEW PT(  308.47, 290.32, 135.00); -- componente2

-- PONTOS INTERMEDIARIOS PARA ALIMENTADOR E
PLACAS

WT1_3  : NEW PT(  14.10, 401.87, -45.00); -- componente1
WT1_4  : NEW PT(  114.20, 494.75, -45.00); -- componente1
WT1_5  : NEW PT(  161.56, 508.95,  0.00); -- componente1
WT1_9  : NEW PT(  331.24, 325.94, 135.00); -- componente1
WT1_10 : NEW PT(  233.12, 233.84, 135.00); -- componente1
WT1_11 : NEW PT(  185.82, 221.19,-180.00); -- componente1
WS1  : NEW PT(  -85.17, 353.52,-135.22); -- componente1
WT2_7  : NEW PT(  46.17, 439.02, -45.00); -- componente2
WT2_13 : NEW PT(  298.85, 293.05, 135.00); -- componente2
WS2  : NEW PT( -115.69, 407.66,-135.22); -- componente2

-- JA EXISTE UM ALIMENENTADOR PARA O TIPO ABAIXO
-- WS2  : NEW PT( -115.69, 407.66,-135.22); -- componente2
```

```
T1 : NEW < T1_3, T1_4, T1_5, T1_9, T1_10, T1_11 >;

WT1  : NEW  < WT1_3,  WT1_4,  WT1_5,  WT1_9,  WT1_10,
WT1_11 >;

T2 : NEW < T2_7, T2_13 >;

WT2 : NEW < WT2_7, WT2_13 >;

MAIN : SUBR;

  PICK_AND_PLACE : SUBR(DE,PARA,WS,WT);
    PMOVE(WS);
    ZONE(PRECISAO);
    PMOVE(DE);
    DOWN;
    DELAY(ESPERA);
    GRASP;
    DELAY(ESPERA);
    UP;
    ZONE(0);
    PMOVE(WT);
    ZONE(PRECISAO);
    PMOVE(PARA);
    DOWN;
    DELAY(ESPERA);
    RELEASE;
    DELAY(ESPERA);
    UP;
    ZONE(0);
  END;

-- COMANDOS INICIAIS DO ARQUIVO "INITIAL.AML"

UP;
RELEASE;
PMOVE(HOME);
  ITERATE('PICK_AND_PLACE', S1, T1, WS1, WT1);
  ITERATE('PICK_AND_PLACE', S2, T2, WS2, WT2);
END;
```

Figura 5

```
- DEFINICOES DO ARQUIVO DEFAULT.AML :

PRECISAO : NEW 1;
VELOCIDADE : NEW 10;
ESPERA : NEW 1.5;
HOME : NEW PT( 650, 0, 0 );

-- COORDENADAS DOS ALIMENTADORES

S1  : NEW PT(  -93.99, 358.23,-135.22); -- componente1
S2  : NEW PT( -125.31, 410.39,-135.22); -- componente2

-- JA EXISTE UM ALIMENENTADOR PARA O TIPO ABAIXO
-- S2  : NEW PT( -163.64, 448.43,-135.22); -- componente2

-- COORDENADAS DOS COMPONENTES NA PLACA
T1_3  : NEW PT(  -38.00, 575.76, -90.00); -- componente1
T1_4  : NEW PT(  101.87, 573.11,   0.00); -- componente1
T1_5  : NEW PT(  145.30, 548.99,  45.00); -- componente1
T1_9  : NEW PT(  132.57, 293.94,-180.00); -- componente1
T1_10 : NEW PT(   -4.51, 296.59,-180.00); -- componente1
T1_11 : NEW PT(  -47.94, 320.71,-135.00); -- componente1
T2_6  : NEW PT(  -12.31, 577.26,   0.00); -- componente2
T2_13 : NEW PT(   93.27, 292.44,-180.00); -- componente2

-- PONTOS INTERMEDIARIOS PARA ALIMENTADOR E
PLACAS

WT1_3  : NEW PT(  -30.64, 582.53, -90.00); -- componente1
WT1_4  : NEW PT(   97.31, 564.21,   0.00); -- componente1
WT1_5  : NEW PT(  138.31, 541.84,  45.00); -- componente1
WT1_9  : NEW PT(  122.97, 296.74,-180.00); -- componente1
WT1_10 : NEW PT(  -12.23, 302.95,-180.00); -- componente1
WT1_11 : NEW PT(  -54.80, 327.99,-135.00); -- componente1
WS1  : NEW PT(  -87.13, 350.96,-135.22); -- componente1
WT2_6  : NEW PT(  -13.25, 567.30,   0.00); -- componente2
WT2_13 : NEW PT(   84.69, 297.58,-180.00); -- componente2
WS2  : NEW PT( -116.73, 405.26,-135.22); -- componente2

- JA EXISTE UM ALIMENENTADOR PARA O TIPO ABAIXO
-- WS2  : NEW PT( -116.73, 405.26,-135.22); -- componente2
```

```
-MAIN : SUBR;

  PICK_AND_PLACE : SUBR(DE,PARA,WS,WT);
    PMOVE(WS);
    ZONE(PRECISAO);
    PMOVE(DE);
    DOWN;
    DELAY(ESPERA);
    GRASP;
    DELAY(ESPERA);
    UP;
    ZONE(0);
    PMOVE(WT);
    ZONE(PRECISAO);
    PMOVE(PARA);
    DOWN;
    DELAY(ESPERA);
    RELEASE;
    DELAY(ESPERA);
    UP;
    ZONE(0);
  END;

-- COMANDOS INICIAIS DO ARQUIVO "INITIAL.AML"

UP;
RELEASE;
PMOVE(HOME);
  PICK_AND_PLACE( S2 , T2_6, WS2, WT2_6 );
  PICK_AND_PLACE( S1 , T1_3, WS1, WT1_3 );
  PICK_AND_PLACE( S1 , T1_4, WS1, WT1_4 );
  PICK_AND_PLACE( S1 , T1_5, WS1, WT1_5 );
  PICK_AND_PLACE( S2 , T2_13, WS2, WT2_13 );
  PICK_AND_PLACE( S1 , T1_9, WS1, WT1_9 );
  PICK_AND_PLACE( S1 , T1_10, WS1, WT1_10 );
  PICK_AND_PLACE( S1 , T1_11, WS1, WT1_11 );
END;
```

Figura 6

COMENTÁRIOS

O modelamento geométrico é limitado a 2D porque o robô não é servo-assistido no eixo Z, sendo que este eixo é programado apenas para as posições alta ("UP") e baixa ("DOWN").

Uma questão importante é a precisão envolvida (Radhakrishnan 1992), desde que os desenhos CAD são uma representação matemática idealizada da realidade, e temos dificuldade em fazer com que a montagem real coincida com a montagem do lay-out descrito no arquivo CAD.

Além disso temos que levar em conta as caraterísticas de precisão do robô (acurácia e repetibilidade), e compará-las com a exigência de precisão da montagem em questão, e determinarmos se a montagem automática é viável. As caraterísticas do robô podem ser determinadas com testes como os propostos por Edkins e Smith (Edkins M. and Smith C. R. T. 1985), e usar métodos de comparação (Vetorazzi Jr., C. N., 1994) para determinarmos a viabilidade.

No mínimo temos uma boa primeira aproximação tanto do programa quanto das coordenadas, o que pode economizar bastante tempo de programação.

REFERÊNCIAS

(Aut89) "AutoCAD Reference Manual", Autodesk Inc., 1989.

(duF86) du Feu, P.H., "FMS for PCB Assembly", Stanford FMS Proceedings, pp 657-666,Stanford , U.K., 1986.

(Edk85) Edkins, M. and Smith, C. R. T., "The Practical Problems Involved in off-line Programming a Robot from a CAD System", in Robots and Automated

Manufacture, edited by J. Billingsley, P. Peregrinus Ltd., pp 29-39, London, U. K., 1985.

(Jac91) Jacobs, F. et al, "A Rule Based System to Generate NC Programs from CAD Exchange Files", Comp. Ind. Eng., V 20, n. 2, pp 214-3224, 1991.

(Sup91) Supinski, M.R. et al, "Automatic Plan and Robot Code Generation for PCB Assembly", Manufacturing Review V. 4, n. 3, pp 214-224,1991.

(Rad92) Radhakrishnan,T.,"Combined Efects of Linear and Angular Errors in PC Board Assembly", Int. Journal of of Production Research, V. 30, n. 5, pp 1037-1044, 1992.

(Ve94a) Vetorazzi Jr, C. N.,Telles,G. N.,"Robot Program Generation for PCB (Printed Circuit Board) Component Insertion Via CAD System",IEEE ISIE 94 Proceedings,Santiago, Chile, 1994.

(Ve94b) Vetorazzi Jr.,C.N., "Geração Automática de Programas para um Robô Industrial", Dissertação de Mestrado, Faculdade de Engenharia Mecânica - UNICAMP , 1994.

O PARADIGMA DE OBJETO E ELEMENTOS DE INTELIGÊNCIA ARTIFICIAL PARA GERAÇÃO AUTOMÁTICA DE CÓDIGO DE CONTROLE

OBJECT-ORIENTED PARADIGM AND ARTIFICIAL INTELLIGENCE ELEMENTS FOR CONTROL CODE AUTOMATIC GENERATION

Prof. Dr. Antonio BATOCCHIO*
Orlando Durán ACEVEDO**

ABSTRACT

This paper presents a manufacturing system specification technique that combines Logic Grammars and the Object-Oriented Paradigm. As an application, an automatic PLCs program generator which uses the object oriented specification technique was created. Main concepts of the Object-Oriented Paradigm are discussed and the use of logic grammars and their applications in lexical and syntactical analysis are also commented.

KEY WORDS: Object-Oriented Paradigm, Logic Grammars, Manufacturing Systems, PLCs, Artificial Intelligence.

RESUMO

Este trabalho apresenta uma metodologia que combina Gramáticas Lógicas e o uso do Paradigma de Orientação a Objeto para criação de uma técnica de especificação, orientada a objeto, de sistemas de manufatura. Como uma aplicação desta técnica é descrito um sistema de geração automática de programas para Controladores Lógicos Programáveis (CLPs). São apresentados os conceitos fundamentais do Paradigma de Orientação a Objeto como também são discutidas as Gramáticas Lógicas, baseadas nas Gramáticas de Cláusulas Definidas (Definite Clause Grammars-DCGs), usadas aqui para as tarefas de análise léxica e sintática da linguagem de descrição da lógica de controle, como também na geração automática do código fonte escrito no formato específico de um CLP.

PALAVRAS-CHAVE: Paradigma de Objeto, Gramáticas Lógicas, Sistemas de Manufatura, CLPs, Inteligência Artificial.

INTRODUÇÃO

Este trabalho enfoca a geração automática de software de controle para Células Automatizadas de Manufatura (Automated Manufacturing Cells - AMC), usando Controladores Lógicos Programáveis (CLPs). O objetivo deste projeto é construir um sistema para gerar programas automaticamente e controlar ou gerenciar as operações no interior de uma AMC. Esse sistema utiliza, como idéia central, os conceitos do Paradigma de Orientação a Objeto para fornecer um meio eficaz para a especificação da estratégia de controle que será usada na coordenação das tarefas da Célula. Com esses conceitos o usuário pode fazer uso de uma coleção de classes de objetos e de relações entre eles para descrever os eventos que serão levados a cabo durante a operação da Célula.

A tarefa de geração utiliza-se de elementos de Inteligência Artificial, tais como as Gramáticas de Cláusulas Definidas (Definite Clause Grammars - DCGs) para realizar o processo de análise da Especificação Objeto-Orientada ("scanning") e para a geração do código propriamente dita.

(*) Dr. Antonio Batocchio - Depto. Eng. de Fabricação/Faculdade de Eng. Mecânica UNICAMP.
(**) MSc. Orlando Durán Acevedo - Depto. Eng. de Fabricação/Faculdade de Eng. Mecânica UNICAMP.

Neste trabalho são discutidos, primeiro, o Paradigma de Orientação a Objeto como idéia central e, segundo, o uso das Gramáticas de Cláusulas Definidas. Também apresenta-se um protótipo escrito em Arity Prolog para a geração de programas para CLPs, e finalmente são enumeradas as possibilidades para trabalhos futuros e otimizações no uso da técnica.

O PARADIGMA DE ORIENTAÇÃO A OBJETO

Como Cox e Novobilski [COX-91] comentaram, desafortunamente a indústria do software possui um "record" negativo no fornecimento dos seus produtos. Os sintomas fatais, desgraçadamente, nos parecem familiares: erro no planejamento, falta total de controle das operações, baixa qualidade, etc. Esses problemas podem derivar, eventualmente, no cancelamento do projeto e no descrédito para o fornecedor. Em ambientes tão competitivos, que atualmente se apresentam, isto pode causar um efeito mortal para uma empresa.

O problema real é a incapacidade dos produtores de software de enfrentar e responder a uma série de situações que atentam contra a qualidade de produto e do serviço que estes fornecem. Entre estas situações pode-se citar: as mudanças nos requerimentos, alterações do projeto, descrição ambígua das necessidades, etc. Dito isto, pode-se concluir que existe a necessidade da criação de novas ferramentas de desenvolvimento e metodologias para melhorar a tarefa de construção de software. O paradigma de orientação a objeto, aplicado tanto nas fases de análise como na fase de programação, parece a técnica mais adequada com as tendências atuais do mercado de software. Isto pode ser ampliado para as tarefas de geração de software de controle e, em geral, na produção de aplicações de gerenciamento de processos automatizados.

Do paradigma de orientação a objeto à diferença das metodologias de análise e programação tradicionais (com uma visão funcional da realidade), há uma divisão do domínio num conjunto de classes de objetos. Este conceito, usado inicialmente na Engenharia de Software [WASS-90], causou mudanças revolucionárias no projeto e escrita de software.

Atualmente, a indústria de software procura nas suas operações, elementos importantes, tais como produtividade, confiabilidade, reutilização e fácil manutenção; para isto, as novas plataformas de desenvolvimento de software estão incorporando os conceitos da orientação a objeto. Conseqüentemente, um conjunto de novas linguagens que usam esse paradigma, tem se incorporado ao mercado. Alguns exemplos são SmallTalk, ADA e C++. A seguir, são comentados os conceitos principais do paradigma de orientação a objeto.

O conceito de software orientado a objeto foi chamado por Cox [COX91] de Integrated Circuit Software (software ICs) para enfatizar a semelhança existente entre estes módulos de software e os chip de circuitos integrados. Assim, um programador não precisa mais construir programas inteiros a partir do nada, agora ele pode construir novos programas através da montagem de componentes de software reutilizáveis de outros programadores (FIGURA 1).

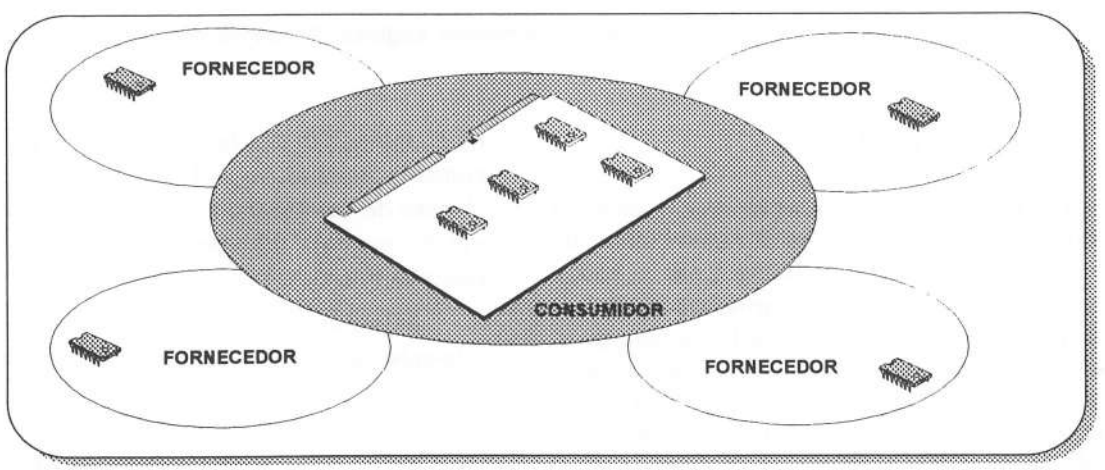


Figura 1 - O usuário pode montar as suas aplicações a partir de um conjunto de componentes adquiridos de diferentes fornecedores.

A tendência é que a construção de software se torne cada vez mais semelhante às linhas de montagem industriais, onde os montadores apenas compram componentes no mercado e os combinam, de maneira tal, a obter um produto.

CONCEITOS

Antes de dar continuidade, deve-se entender o conceito de OBJETO. Objeto é uma entidade a qual encapsula dados (atributos) e procedimentos (métodos). Qualquer objeto do mundo real pode ser visto como um objeto e, portanto, qualquer objeto pode ser representado como uma peça de software. O processo de representação de um objeto do mundo real numa peça de software na memória de um computador ou num modelo orientado a objeto, é chamado de **ABSTRAÇÃO**. Assim, através da abstração, a percepção de um objeto (no modelo) será muito próxima à percepção do objeto real. Isto é, para cada objeto considerado do mundo real, o modelo associa um conjunto de atributos e métodos, os quais são inerentes ao objeto do mundo real [BOOC-86]. O processo de abstração é esquematizado na figura 2.

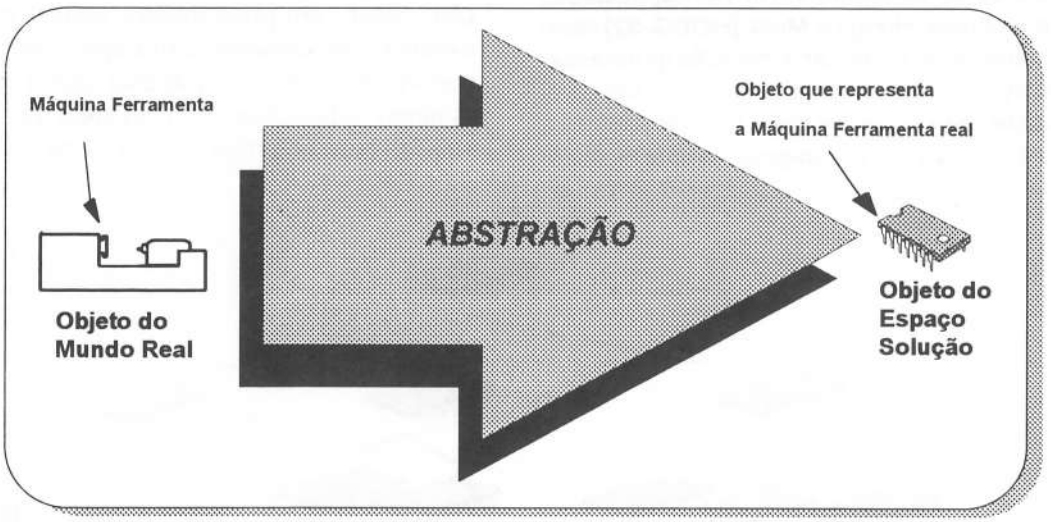


Figura 2 - O conceito de abstração. O programador visualiza uma entidade do mundo real e logo cria uma representação simbólica no espaço solução. Esta representação é chamada de "objeto".

Encapsulamento é provavelmente o conceito mais importante, sobre o qual o paradigma se baseia. Como mencionado anteriormente, cada objeto é associado a um conjunto de operações e atributos; esses atributos apenas podem ser acessados pelas operações definidas no interior do objeto. Usualmente, essas operações estão localizadas numa parte privada do objeto e são acessadas através de mensagens. A identificação dos métodos e atributos de um objeto é feita numa seção pública chamada de interface, através da qual as mensagens passam para o "interior do objeto". Cox e Novobilski [COX-91] descrevem este conceito como "uma muralha de código ao redor de uma estrutura de dados" (FIGURA 3)

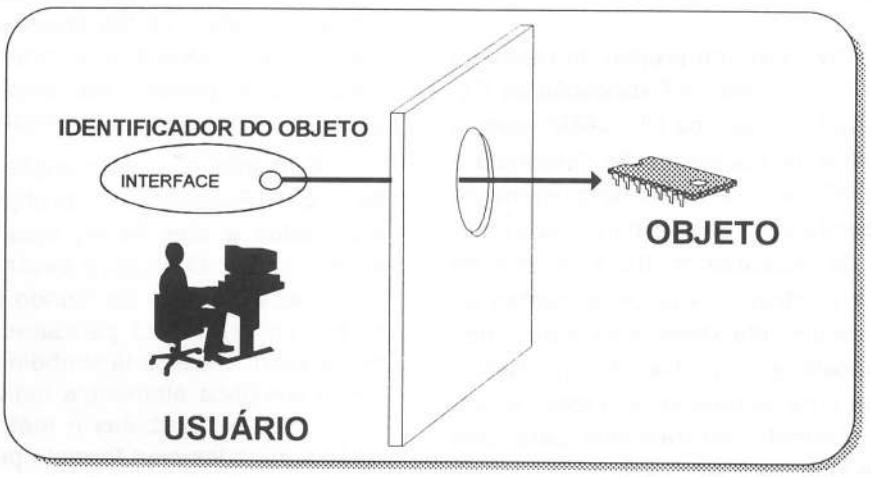


Figura 3 - O Conceito de encapsulamento. Existe uma muralha rodeando cada objeto. O usuário apenas especifica o que ele deseja que o objeto faça através do envio de uma mensagem. A forma como o objeto realiza a ação desejada, ficará oculta numa parte privada do objeto.

Através deste conceito os detalhes da implementação são escondidos numa caixa preta, a seção privada do objeto. Assim, o usuário do objeto pode especificar apenas o que o objeto deve realizar, deixando ao objeto a tarefa de escolher o código que é mais apropriado para o tipo ou classe.

O terceiro conceito é a **HERANÇA**. Esta é a capacidade de construção de hierarquias de objetos. Em forma similar a uma árvore genealógica, vários objetos podem ser originados a partir de um objeto comum ou classe comum. Conseqüentemente, os objetos herdados podem ser extendidos, incorporando novas capacidades e adicionando atributos. Hodge e Mock [HODG-92] identificam esse mecanismo como uma "relação generalização-especialização", onde a superclasse é a versão geral de uma subclasse mais especializada. Esse conceito é o aspecto mais inovativo do paradigma, já que a maior parte das técnicas tradicionais (funcionais) não incorporam a capacidade de gerar subclasses a partir de uma superclasse existente.

A capacidade de herdar atributos e operações fornece as ferramentas para reutilizar blocos de software. Isso permite ao programador ou analista definir um novo software, ou novo modelo, da mesma forma que um professor (ou especialista) introduz um conceito ao leigo (ou aprendiz), através da comparação com alguma situação similar, que resulta familiar e que, com pequenas modificações, poderá ser extendida para a nova situação. Assim, um programador poderá pegar uma peça existente de software com objetos em comum com o domínio do problema que está sendo resolvido, e com mínimas alterações e/ou adições poderá gerar uma solução para tal problema (FIGURA 4)

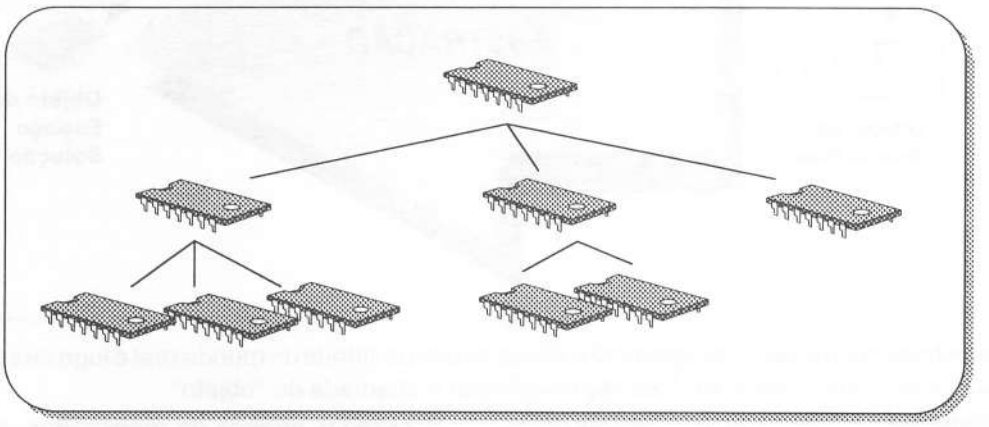


Figura 4 - O conceito de herança. Uma classe primária origina diferentes subclasses. Essas subclasses podem herdar algumas caraterísticas das superclasses e, adicionalmente, incorporar outras caraterísticas próprias.

A TÉCNICA DE ESPECIFICAÇÃO ORIENTADA A OBJETO

Está sendo desenvolvido um projeto de pesquisa no Departamento de Engenharia de Fabricação da Faculdade de Engenharia Mecânica da UNICAMP, para a criação de uma Técnica de Especificação Orientada a Objeto (TEOO). TEOO servirá como linguagem de modelamento ou especificação da estrutura e funcionamento de sistemas de manufatura. Essa técnica de especificação agrupa os dispositivos pertencentes ao domínio da manufatura discreta, detecta os seus principais atributos e capacidades e os traduz em objetos. Dessa forma, criou-se uma hierarquia com estes dispositivos e que, posteriormente, foi traduzida para uma árvore de Classes de Objetos [FAPE-93].

Todas as classes de objetos definidas foram herdadas de uma classe comum, DISPOSITIVO. Abaixo dessa classe outras classes foram herdadas. Essas classes foram, Robô Industrial, Máquinas Ferramenta, Dispositivos de Manuseio de Materiais, etc. Posteriormente um conjunto de procedimentos foram atribuídos a cada uma dessas novas classes. Adicionalmente cada dispositivo pode ter um conjunto de atributos particulares.

Com toda essa informação, foi criado um conjunto de objetos. Esse conjunto de objetos e os procedimentos associados a eles foram incorporados à técnica de especificação. O objetivo dessa especificação é representar as instâncias do mundo real através de objetos pertencentes à TEOO, para serem usadas nas tarefas de modelagem. Para cada símbolo na especificação o processo identifica elementos individuais do modelo tais como, objetos, atributos e métodos. Esses elementos estão agrupados num formato pré-definido. A forma em que esses elementos estão agrupados e que podem ser combinados está regida por em conjunto de regras. Este conjunto de regras é chamado de Gramática.

A TÉCNICA DE GRAMÁTICAS DE CLÁUSULAS DEFINIDAS

Como mencionado anteriormente, uma gramática é um conjunto de regras que governam as possibilidades combinatoriais dos símbolos básicos (primitivas) de uma linguagem. Nesta seção são apresentados os conceitos fundamentais da construção de gramáticas, e a sua aplicação na construção de uma técnica de especificação e, posteriormente, a sua aplicação num sistema de geração automática de software para CLPs.

Existem diferentes tipos de gramáticas. Chomsky [CHOM-56] relatou quatro tipos de gramáticas. A maioria das linguagens de programação pode ser considerada como pertencendo ao tipo de gramática chamada Gramáticas de Livre Contexto (Context Free Grammar - CFG). Nas CFG as palavras de uma linguagem são chamadas de símbolos terminais e as estruturas de maior complexidade (os símbolos não-terminais) podem ser decompostas numa combinação de símbolos terminais seguida, eventualmente, de outros símbolos não-terminais. Cada regra de uma CFG mostra como um símbolo não terminal pode ser decomposto numa combinação de símbolos terminais e/ou outros símbolos não-terminais. A seguir é mostrada uma gramática deste tipo. A representação usada é chamada de Cackus Naur Form.

program ::= <statement><program >
program ::= []
statement ::= [let],[id(v)] [:=] <expression>
statement ::= [if]<condition>[then] program [endif]
condition ::= [not]<relation>
condition ::= <relation>
relation ::= <expression> <comp_op><expr>
comp_op ::= ['=']
comp_op ::= ['<']
expression ::= <primitive>
expression ::= <expression> <arit_op> <primitive>
primitive ::= [id(V)]
primitive ::= [num(N)]
arit_op ::= ['+']
arit_op ::= ['-']
arit_op ::= ['*']
arit_op ::= ['/']

A primeira regra indica que um programa é formado por um **statement** mais um **program** (no resto dos **statement** note o caráter recursivo), ou eventualmente, um programa vazio, indicado pelos colchetes**[]**. Um **statement** pode ser um comando do tipo LET. Este comando está definido através da terceira regra que combina o símbolo terminal **let**, indicado dentro dos colchetes**[]**, mais um identificador de variável, indicado pelo símbolo [id(V)], mais o símbolo terminal'=', também entre colchetes '[]', mais uma expressão, indicada pelo símbolo não-terminal **<expression>**, a qual possui a regra, no caso duas regras, correspondentes.

Estas CFGs não são totalmente apropriadas para descrever a linguagem natural e algumas classes de Linguagens de Programação existentes. Assim, Colmerauer e Kowalski criaram um outro formalismo, chamado de Gramática de Cláusulas Definidas (Definite Clause Grammar, DCGs), mais aptas para o tratamento das regras da gramática [COLM-77]. Esses princípios originaram a criação da linguagem Prolog. Prolog pode representar um conjunto de cláusulas (no caso, regras da gramática) num programa escrito em lógica. Essa técnica permite escrever gramáticas de uma forma muito facilitada e natural. Através do uso de DCGs podem ser construídos analisadores sintáticos, interpretadores, e até compiladores para linguagens de programação [SZPA-87].

O princípio usado para construir o sistema de programação automática de CLPs é uma combinação entre as DCGs e as chamadas de "Case Grammar", ou Gramáticas de Caso, onde cada símbolo de técnica pode ser associado a uma categoria dentro da descrição. As categorias usadas neste trabalho foram especificadas por Su em [SU-89]:

- Especificação de partes ou facilidades de um sistema.

- Descrição de um evento ou atividade.

- Especificação de valores de atributos de eventos, atividades partes ou facilidades.

- Definição do estado de facilidades ou partes, o que corresponde ao acontecimento de um evento.

Os objetos definidos foram adicionados à gramática na forma de símbolos terminais (chamados também de palavras reservadas). Isto é, os objetos, os seus atributos e os identificadores dos procedimentos foram incorporados ao vocabulário da gramática. Posteriormente, todos esses procedimentos foram incorporados ao vocabulário da gramática descrita. Algumas dessas categorias são formadas por combinações de elementos como por exemplo, uma atividade é a especificação de um objeto mais um procedimento associado a ele. Finalmente, a técnica de especificação é estruturada usando um conjunto de sentenças com uma topologia pré-definida, que é regida pelas categorias já citadas.

Como aplicação da TEOO, foi criado um protótipo de programação automática para CLPs [DURA-94]. Esse sistema gera, a partir da especificação da estratégia de

controle escrita em TEOO, programas escritos num subconjunto da linguagem de programação para Controladores da Allen Bradley, chamado PLC4. A estrutura desse sistema é mostrada na figura 5.

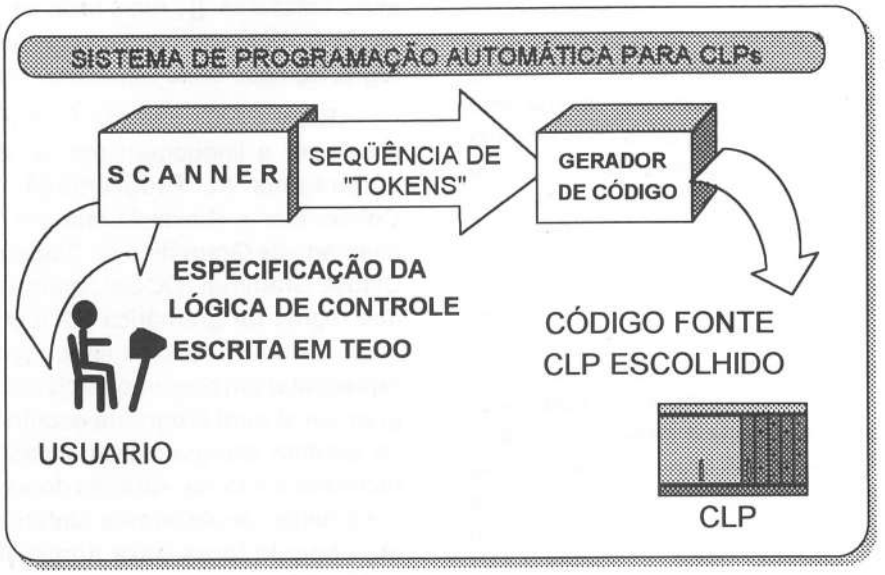


Figura 5 - Estrutura do sistema de geração automática de software para CLPs a partir de uma descrição feita em TEOO.

O primeiro módulo, o "scaner", realiza a análise léxica e sintática da descrição em TEOO. Isto é feito confrontando a descrição com a gramática da TEOO. A saída desse módulo é uma representação "tokenizada" das sentenças que compõem a descrição Essa representação "tokenizada" é transferida para o módulo gerador de software. Este é encarregado de transformar a representação "tokenizada" para formato de programação do CLP escolhido. Este último módulo é a única peça dependente do CLP escolhido, ou seja, existirá um módulo para cada CLP que o usuário quiser incorporar ao sistema. Na figura 6, esse conceito é mostrado. A área encerrada pelo retângulo tracejado representa a única parte do sistema que depende do CLP selecionado para ser programado.

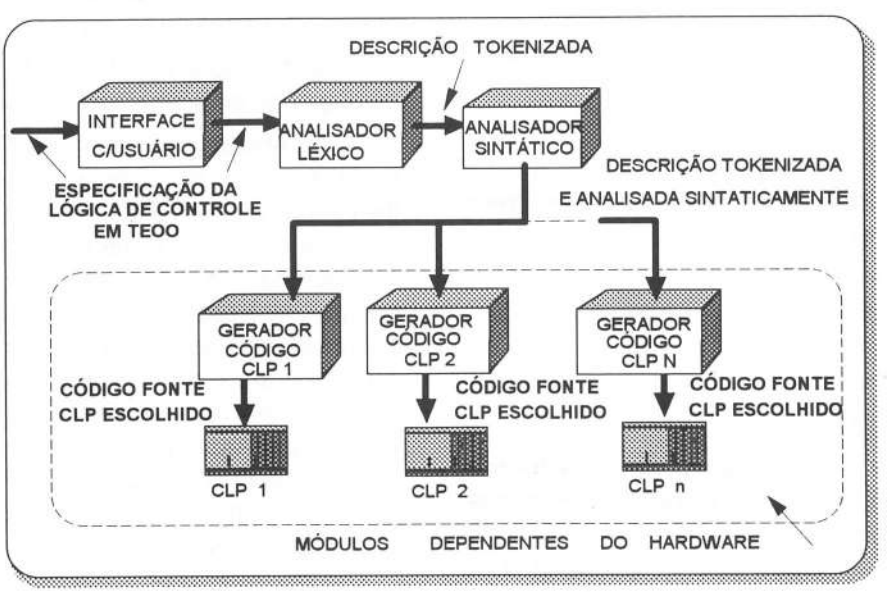


Figura 6 - Módulos do sistema de programação automática para CLPs

POR QUE OBJETOS?

TEOO pode ser usada para a criação de bibliotecas de descrições que armazenam informação técnica e capturam conhecimento sobre experiências prévias. Esse conhecimento arquivado será de grande utilidade na tarefa de implementação de novos sistemas de manufatura.

A mesma filosofia pode ser usada no desenvolvimento de software de controle onde um novo programa pode ser gerado a partir de poucas alterações feitas num programa existente. A semelhança de uma situação com outra pode ser melhor entendida se a linguagem usada para descrever a estratégia de controle é uma linguagem de alto nível. Assim a reutilização de código é o principal atrativo da TEOO.

Com a criação de bibliotecas de objetos e arquivos de especificações escritas usando a TEOO, a reutilização do software de controle criado usando o paradigma de objeto está garantida. Junto com isto, o conceito de herança permite a rápida prototipação de novos programas para gerenciamento de uma célula de trabalho. Isso é feito através da modificação de um programa existente que implementa uma solução para um problema análogo.

A estrutura da Técnica de Especificação Orientada a Objeto é facilmente implementada de acordo com as DCGs, o que permite a incorporação dinâmica de novas classes e instâncias de objetos de uma forma muito natural.

Freqüentemente existem diferentes tipos de especialistas e técnicos trabalhando juntos num projeto de automação. Assim, pode-se encontrar eletrônicos, eletricistas e engenheiros mecânicos entre outros. Esses profissionais não usam a mesma linguagem e ferramentas para descrever os projetos. Essa situação leva a problemas de comunicação e duplicidade de informação, o que pode afetar consideravelmente a produtividade de tais profissionais. TEOO pode ser usada como uma linguagem de alto nível para ajudar na comunicação entre esses diferentes tipos de especialistas, dando a flexibilidade suficiente a um para se comunicar com os outros.

Finalmente, num ambiente industrial, existe um conjunto heterogêneo de controladores, e cada um incorpora linguagens de programação proprietárias. Por este motivo, as possibilidades de migração entre um sistema e outro resultam muito caras [ROBE-93]. Assim a escolha mais freqüente é o desenvolvimento do novo programa a partir de zero ao invés de alterar algum programa existente (processo variante). Se um fabricante usar essa técnica e descrever todas as suas estratégias de controle usando TEOO, ele poderá transferir programas de um controlador para outro, simplesmente executando a geração automática do programa, usando como entrada a especificação objeto orientada.

CONCLUSÕES

Está em desenvolvimento um sistema de geração de software de controle. Este sistema usa como entrada uma Técnica de Especificação Orientada a Objeto. Essa TEOO é interpretada pelo sistema, usando a metodologia de DCGs e, posteriormente, traduzida para um programa de controle escrito num formato específico para um CLP. No estágio atual, o sistema roda em ambiente DOS, e realiza a geração de código escrito num subconjunto da linguagem de controle PLC4, a qual a Allen Bradley incorpora à maioria dos seus controladores.

Melhoramentos futuros apontam a construção de uma versão que seja capaz de rodar em ambiente Windows, como também a incorporação de outras linguagens de programação para testar o conceito de geração automática de código a partir da TEOO.

AGRADECIMENTOS

Este trabalho está sendo desenvolvido com o apoio financeiro da Fundação de Amparo à Pesquisa do Estado de São Paulo, FAPESP, e o Fundo de Apoio ao Ensino e Pesquisa, FAEP/UNICAMP.

REFERÊNCIAS

[BOOC-86] Booch, G., "Object Oriented Design", IEEE Transactions on Software Engineering, Vol. SE-12, n. 2, pp. 27 - 35, 1986.

[CHOM-56] Chomsky, N. "Three Models for the Descriptions of Languages". IEEE Transactions on Information Theory, v. IT-2, pp 113-124, 1956.

[COLM-77] Colmerauer, A. "An interesting Natural Language Subset" Groupe d'Inteligence Artificielle, Université de Marseille - Luminy, 1977.

[COX-91] Cox, B. J. and A. J. Novobilski. "Object Oriented Programming", 2nd ed, Addison Wesley Pub. Co., USA, 1991.

[DURA-94] Duran, O. M. e A. Batocchio, "A High-Level Object-Oriented Programmable Controller Programming Interface", Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'94), pp 226-230. Santiago, Chile, May, 1994.

[FAPE-94] Duran, O. M., "Relatorio de Atividades Projeto: Tecnica de Especificação de Sistemas de Manufatura (Um enfoque orientado a objeto)". FAPESP 92/4983-0, São Paulo, Agosto, 94.

[HODG-92] Hodge, L. R. and M. T. Mock. "A proposed Object-Oriented development methodology", Software Engineering Journal, March, pp 119-129, 1992.

[ROBE-93] Roberts, C. A. and T. G. Beaumariage, "A Specification Techinique for Generating and Simulating Supervisory Control", Computers and Industrial Engineering, vol 25, nos. 1-4, pp 515-518, 1993.

[SU-89] Su, H.M. and V. Kachitvichyanukul, "A Natural Language System to aid Simulation Model Formulation", Computers and Industrial Engineering, vol 16, n 4, pp 535-543, 1989.

[SZPA-87] Szpakowicz, "Logic Grammars", Byte, August, pp 185-195, 1987.

[WASS-90] Wasserman, A. I., P. A. Pircher and R. J. Muller, "The Object Oriented Structured Design Notation for Software Design Representation", Computer, vol 23, n. 3, pp 50-63, 1990.

OPINIÃO

O MITO DO BACKLOG

THE BACKLOG MYTH

Prof. José Estevão PICARELLI*

ABSTRACT

The term know as "backlog", applied a lot for justifying the information support expansion (PEOPLEWARE, HARDWARE AND SOFTWARE) in the centralized units of information systems (DATA CENTER, INFORMATION DEPARTMENT, SYSTEMS DIVISION), reveals for true an administrative deficiency concerning the planning of its services. This task presents 4 basic reasons which gives rise to the "backlog" formation and a model typifying the various states in search of information service for assistance in planning process.

RESUMO

O chamado "*backlog*" (demanda reprimida), muito utilizado como justificativa para expansão de recursos de informática (PEOPLEWARE, HARDWARE E SOFTWARE) nas unidades centralizadas de sistemas de informação (CPD, DEPARTAMENTO DE INFORMÁTICA, DIVISÃO DE SISTEMAS) reflete na verdade uma deficiência administrativa no que diz respeito ao planejamento de seus serviços. Neste trabalho são apresentadas 4 razões básicas que dão origem à formação do "*backlog*" e um modelo representando os diversos estados da demanda de serviço de informática para auxilio à um processo de planejamento.

1. INTRODUÇÃO

No contexto de informática, o termo "*backlog*" é usado para denotar o conjunto de solicitações de serviços, principalmente solicitações de desenvolvimento de sistemas de software, que estão em uma fila de espera para, quando existir disponibilidade de recursos, serem, um dia, atendidas.

Na tentativa de eliminar, ou pelo menos diminuir, o tamanho dessa fila, falta de recursos humanos, falta de ferramentas automatizadas e baixa produtividade são julgados como os culpados da formação dessa fila e o suprimento de todos ou de alguns desses itens são apontados como soluções para o problema.

Nessa linha, quando essas supostas soluções são implementadas, contrariamente ao esperado, o "*backlog*" aumenta.

Outro aspecto interessante é observarmos onde existe e onde não existe o "*backlog*". Por exemplo, nas chamadas "SOFTHOUSE", empresas de desenvolvimento de software, não existe a formação do "*backlog*". Alguns empresários dessa atividade afirmam que até gostariam de ter o "*backlog*", dando a entender que, assim, possuiriam uma boa carteira de clientes e pedidos de serviços. Porém, quando isso ocorre, nasce nos clientes uma crescente insatisfação, que provoca o fim do "*backlog*" com o fim da própria empresa. Tudo indica, portanto, que o "*backlog*" é privilégio de empresas, de caráter público ou privado, onde a unidade de sistemas de informação é centralizada e exerce um papel de assessoria.

O objetivo deste trabalho é mostrar as razões básicas da formação do "*backlog*", para que, com esse conhecimento, possamos evitá-lo.

(*) Prof. José Estevão Picarelli - Mestrando do Curso de Mestrado em Gerenciamento de Sistemas de Informação - Instituto de Informática - PUCCAMP - Professor do Instituto de Informática - PUCCAMP.

2. CAUSAS DA FORMAÇÃO DO *"BACKLOG"*

2.1. OS GERADORES DE DEMANDAS

Os clientes das unidades centralizadas de sistema de informação estão distribuídos pela empresa nos seus diversos departamentos, divisões e setores e são comumente chamados de comunidade usuária ou simplesmente de usuários.

Essa entidade, usuário, é referenciada em normas, metodologias, discursos e manuais de diretrizes, como sendo a principal figura e razão fundamental das atividades e serviços da unidade de informática.

Assim como o próprio nome (usuários) diz, seria esperado que o maior consumo dos recursos de informática e o maior número de solicitações de serviços viessem deles. Porém, na prática, isso não ocorre.

Imbuídos de um entusiasmo tecnológico e de uma apatia informacional, os próprios profissionais de informática são os responsáveis pela geração da grande parte de demandas de serviços de informática.

Tendo como justificativa o avanço tecnológico na área de processamento eletrônico de dados e a dificuldade de manutenção de software já existentes, a maior parte dos recursos são requisitados para reconstruir, em outra tecnologia, o passivo de software da instalação.

2.2. O ORÇAMENTO DO CLIENTE

Quando desejamos algum produto ou serviço e assumimos o papel de cliente perante algum fornecedor, sabemos que não podemos efetivar nossas solicitações baseados apenas em nossos desejos ou justas necessidades. Existe um outro item, talvez o mais importante, que seria a nossa capacidade de pagamento, ou seja, baseamos nossas solicitações conforme o nosso ORÇAMENTO.

Nas organizações onde a unidade de sistemas de informação é centralizada, na relação cliente (usuário) e fornecedor (informática), isso não ocorre. O cliente sente-se no direito de solicitar o que julgar necessário. O cliente possui ORÇAMENTO ilimitado, ou seja, suas demandas são solicitadas baseadas no ORÇAMENTO do fornecedor. É evidente que, dessa forma, a força de trabalho do fornecedor, independentemente de que tamanho seja, se esgota rapidamente, provocando a criação de uma fila de espera chamada de "*backlog*".

2.3. O ESFORÇO GASTO NAS UNIDADES DE SISTEMAS DE INFORMAÇÃO

Segundo estatísticas, um bom percentual, (60% a 80%) da força de trabalho em um ambiente de desenvolvimento de software está permanentemente alocado em atividades chamadas de manutenção. Para a tecnologia de software esse termo é, no mínimo, inadequado já que, como software não gasta, não precisaria ser mantido.

O trabalho realmente efetuado sob o titulo de manutenção, consiste, em sua maior parte, na remoção de defeitos do software e, em sua menor parte, na inclusão de novas funções ao software já existente.

Seria portanto interessante que, para liberar recursos para minimizar a formação do "*backlog*", as discussões fossem direcionadas mais para o tema da qualidade em vez de produtividade.

2.3. PLANEJAMENTO BASEADO EM DEMANDAS SOLICITADAS

As demandas de serviços de informática são adjetivadas em apenas quatro estados: solicitada, em andamento, cancelada e concluída.

Os planejamentos efetuados com apenas essas visões dos estados da demanda, ficam excessivamente distantes, após algum tempo, do realizado, e às vezes, para justificar essa diferença, é criado mais um estado de demanda: a demanda reprimida ("*backlog*").

A seguir, é apresentado um modelo de estados da demanda de serviços de informática referenciados pelos números de 01 a 14 e os números que aparecem entre parênteses significam os possíveis estados destinos.

01. DEMANDAS DESCONHECIDAS (03)

São as que têm origem no ambiente fora da empresa. Mudanças de legislação, alteração de moeda, concordata de um fornecedor, são exemplos de eventos que podem provocar o nascimento de demandas de serviço de informática para a organização.

Como auxilio à previsão dessas demandas, podem ser utilizados os diversos sistemas gerais de comunicação: rádio, TV, jornais e outros do tipo.

02. DEMANDAS INVISÍVEIS (03)

Representam o volume de trabalho necessário para remover os defeitos que existem no software e que ainda não foram detectados.

03. DEMANDAS VISÍVEIS (04)

Representam o volume de trabalho para transformar as soluções provisórias, que foram dadas em circunstâncias emergenciais, em soluções permanentes.

04. DEMANDAS SOLICITADAS (05,06,12)

São as registradas, geralmente em um documento formal, e que normalmente acarretam em um novo trabalho de desenvolvimento de sistemas.

05. DEMANDAS REJEITADAS POR INCOMPLETEZA

São as demandas solicitadas onde o problema ou necessidade carece de dados para seu completo entendimento. Relatam mais sugestões de solução segundo a visão do usuário do que o problema em si.

06. DEMANDAS CARACTERIZADAS (07,08,11,12)

São as demandas classificadas segundo critérios administrativos adequados à instalação. Necessidade de adaptação, necessidade de correção, aperfeiçoamento, melhoria desejável e melhoria imperativa podem ser exemplos dessa classificação.

07. DEMANDAS REJEITADAS POR NÃO TEREM JUSTIFICATIVA INFORMACIONAL

São as demandas que possuem balanço negativo no resultado da análise custo beneficio.

08. DEMANDAS AVALIADAS (09,10,11)

São as demandas mensuradas (QUANTIFICADAS) segundo valores de informática. Somente neste estado deve ser dado o "ACEITE" da demanda.

09. DEMANDAS REJEITADAS - INOPORTUNAS (03)

São demandas que, após análise relativa com as demais, não são priorizadas.

10. DEMANDAS PLANEJADAS (11,12,13)

São demandas onde, para sua execução, são alocados recursos humanos e prazos adequados.

11. DEMANDAS CONCLUÍDAS

São demandas que, segundo critérios e requisitos pré-estabelecidos, são consideradas prontas.

12. DEMANDAS CANCELADAS

São demandas que, por várias razões, têm a sua execução permanentemente interrompida.

13. DEMANDAS EM ANDAMENTO (11,12,14)

São as demandas que estão sendo executadas.

14. DEMANDAS SUSPENSAS

São as demandas que, por várias razões, têm a sua execução provisoriamente interrompida.

CONCLUSÃO

A demanda reprimida (**"backlog"**) representa o efeito (e não causa) de problemas de mau planejamento dos serviços de informática nas instalações onde a unidade de sistemas de informação é centralizada. Ela pode ser totalmente evitada atacando-se as causas que provocam a sua formação. O termo, demanda reprimida (**"backlog"**), pode ser substituído por demanda rejeitada, demanda supérflua ou lista ilimitada de sonhos e desejos.

O MESTRADO EM INFORMÁTICA DA PUCCAMP

O Curso de Mestrado em INFORMÁTICA, oferecido com Área de Concentração em **Gerenciamento de Sistemas de Informação**, teve seu inicio em agosto de 1992 e conta atualmente com 44 alunos regularmente matriculados, dos quais 14 já se encontram em orientação de tese, com previsão de término entre o 2º semestre de 1994 e o 1º semestre de 1995 (vide tabela em anexo).

Atendendo a uma demanda altamente reprimida no setor de recursos humanos pós-graduados em gerenciamento de sistemas, o Mestrado em Informática foi instalado em função de três objetivos fundamentais:

- Desenvolver a formação de recursos humanos com a qualificação especial para o fomento à pesquisa científica e ao magistério superior, assim como para o exercício da alta gerência organizacional, na Área da Informática em geral e dos Sistemas de Informação em particular.

- Desenvolver estudos e pesquisas na Área da Informática, visando o avanço dos conhecimentos científicos e tecnológicos necessários ao projeto, desenvolvimento e implementação de Sistemas de Informação, assim como a integração desses conhecimentos com as mais modernas técnicas gerenciais.

- Desenvolver projetos para a aplicação de tecnologias de ponta que levem à otimização do desempenho profissional no Gerenciamento de Sistemas de Informação em corporações e organizações públicas e privadas.

De forma a poder atender esses objetivos, o programa do Mestrado em Informática está embasado em quatro linhas de pesquisa:

- **Tecnologias de Suporte dos Sistemas de Informação**: abrangendo estudos e pesquisas voltados aos recursos tecnológicos de hardware e de software necessários ao planejamento e ao desempenho operacional dos sistemas de informação.

- **Sistema de Informação para o Gerenciamento**: abrangendo estudos e pesquisas ligados à teoria da gestão de sistemas, em particular dos sistemas de informação voltados à tomada de decisões em estruturas organizacionais.

- **Gerenciamento de Sistemas de Informação**: abrangendo estudos e pesquisas ligados às estratégias de gerenciamento de sistemas de informação necessárias à otimização de seu desempenho do ponto de vista do usuário final.

- **Informática, Empresa e Sociedade**: abrangendo estudos voltados à análise crítica das tendências e avanços científicos e tecnológicos na Área da Informática, e seus impactos na sociedade em geral e nas corporações em particular.

Para cumprimento do programa previsto, o atual corpo docente do Mestrado em Informática é formado por oito professores/orientadores:

- **Prof. Dr. Maurício Prates de Campos Filho**
  Doutor em Engenharia pela UNS - Argentina, 1972
  (Metalúrgica Nuclear, Engenharia Metalúrgica e de Materiais, Processos de Fabricação, Automação Industrial e Metodologia da Pesquisa)

- **Prof. Dr. Eduardo Oscar de Campos Chaves**
  Doutor em Filosofia (PhD) pela Universidade de Pittsburgh - USA, 1972
  (Filosofia e História da Educação, Informática Médica, Informática na Educação, Gerenciamento de Sistemas e Multimídia)

- **Prof. Dr. Manuel de Jesus Mendes**
  Doutor em Eletrotécnica pela Universidade Técnica de Berlim - RFA, 1968
  (Controle Adaptativo, Otimização Estocástica, Automação Industrial, Tecnologias de Suporte a Sistemas, Redes e Teleinformática)

- **Prof. Dr. Nelson de Jesús Parada**
  Doutor em Filosofia (PhD) pelo Instituto de Tecnologia de Massachusets - USA, 1968
  (Engenharia Eletrônica, Física do Estado Sólido, Sensoriamento Remoto e Computação Gráfica)

- **Prof. Dr. Silas Marques de Oliveira**
  Doutor em Filosofia (PhD) pela Universidade de Illinois - USA, 1991
  (Gerência de Recursos Humanos, Gestão de Pessoal em Organizações Informatizadas e Técnicas de Gerência de Equipes)

- **Profª Drª Cecília Carmem Cunha Pontes**
  Doutora pela Universidade de São Paulo - USP, 1982
  (Bancos de Dados de Acesso Público, Informatização de Documentação Científica e Acesso a Redes Corporativas)

- **Prof. Dr. Geraldo Nonato Telles**

  Doutor pela Universidade Estadual de Campinas - UNICAMP, 1990

  (Automação Industrial, Robótica, Engenharia de Produção e Engenharia Auxiliada por Computador)

- **Prof. Dr. Juan Manuel Adán Coello**

  Doutor pela Universidade Estadual de Campinas - UNICAMP, 1993
  (Inteligência Artificial, Sistemas Especialistas e Redes Neurais)

Mestrado em Informática - Puccamp
1º semestre de 1994
DISSERTAÇÕES DE TESE EM ANDAMENTO

ALUNO	ORIENTADOR	TEMA DE DISSERTAÇÃO	TÉRMINO PREVISTO	LOCAL DE TRABALHO
Joaquim Estevam de Moraes	Prof. Dr. Maurício Prates	Implementação de Sistema de Informação para Aplicação do Método ABC em Forjaria	2º sem. 1994	Equipamentos CLARK
Orandi Mina Falsarella	Prof. Dr. Eduardo Chaves	Desenvolvimento de Sistema de Apoio à Decisão na Área de Infecção Hospitalar	2º sem. 1994	HC Puccamp
Cristiano Roque Portella	Prof. Dr. Manuel Mendes	Metodologia de Prototipagem para Software Orientado a Objeto	2º sem. 1994	Citro Pectina
Ary Tomáz Gomes Jr.	Prof. Dr. Maurício Prates	Informatização da Documentação de Sistemas de Informação	2º sem. 1994	Informática Municípios Ass.
João Carlos Orosz	Prof. Dr. Manuel Mendes	Estruturas Avançadas de Comunicação em Empresas de Grande Porte	2º sem. 1994	CPFL Campinas
Jorge Luís Cordenonsi	Prof. Dr. Maurício Prates	Planejamento Estratégico de Sistema de Informação Utilizando Reengenharia de Processos	2º sem. 1994	Textil Nova Odessa
José Airton Martins	Prof. Dr. Eduardo Chaves	Sistema de Informação para o Planejamento Estratégico em Organizações Informatizadas	2º sem. 1994	CEF Campinas
Jorge Minoru Shimoda	Prof. Dr. Geraldo Telles	Desenvolvimento de Sistema de Informações para Automação Industrial	1º sem. 1995	Prodome Brasil
Antônio Leão Almeida	Prof. Dr. Silas de Oliveira	Informação e Qualidade no Planejamento Estratégico Organizacional	2º sem. 1994	Banco do Ceará
Cid Evangelista Jr.	Profª Drª Cecília Pontes	Sistema Especialista para Indexação Automatizada de Documentação	2º sem. 1994	CPqD Telebrás
José Estevão Picarelli	Prof. Dr. Maurício Prates	Aplicação de Técnicas de Análise de Sistemas em Modelos de Gerenciamento de Qualidade Total	1º sem. 1995	CPFL Campinas
Edmar Fernandes	Prof. Dr. Maurício Prates	Gerenciamento de Sistemas de Informação Voltados ao Usuário Final	1º sem. 1995	Rigesa Papel e celulose
Wanderley Maciel de Souza	Prof. Dr. Eduardo Chaves	Gerenciamento Estratégico de Sistemas de Informação Voltados à Micro-Informática	1º sem. 1995	Metalúrgica Onça
Fátima Maria Nicolleti	Prof. Dr. Eduardo Chaves	Desenvolvimento de Sistema de Suporte a Decisões Financeiras na Administração Universitária	2º sem. 1994	Centro Comput. UNICAMP

1. APRESENTAÇÃO DOS CURSOS DE ESPECIALIZAÇÃO

CURSOS:

Especialização em Análise de Sistemas

Especialização em Informática

DURAÇÃO:

Os Cursos têm duração de 3 (três) semestres, podendo ser iniciados em março (ou agosto para a especialização em Análise de Sistemas) de cada ano e com término previsto para julho (ou dezembro) do ano posterior. Cada semestre é composto por 15 (quinze) semanas de aula.

CARGA HORÁRIA:

- Especialização em Análise de Sistemas: 360 (trezentos e sessenta) horas, distribuídas em semestres de 120 (cento e vinte) horas.

- Especialização em Informática: 570 (quinhentas e setenta) horas, distribuídas da seguinte maneira: 210 (duzentos e dez) horas no primeiro semestre e 180 (cento e oitenta) horas nos segundo e terceiro semestres.

NÚMERO DE VAGAS:

- Especialização em Análise de Sistemas: 35 (trinta e cinco) horas

- Especialização em Informática: 40 (quarenta) horas

INíCIO DOS CURSOS:

- Especialização em Análise de Sistemas: 1985, tendo sido oferecido a 10 (dez) turmas desde então.

- Especialização em Informática: 1988, tendo sido oferecido 8 (oito) turmas desde então.

EVOLUÇÃO:

Desde sua implantação, os Cursos vêm sendo, sistematicamente, reestruturados, de maneira a serem mantidos atualizados com o estado-da-arte da Informática e com as exigências do mercado de trabalho.

2. OBJETIVO DO CURSO DE ESPECIALIZAÇÃO EM ANÁLISE DE SISTEMAS

O Curso de Especialização em Análise de Sistemas visa a promoção de reciclagem e atualização de profissionais que atuem na área de Análise de Sistemas, de maneira a suprir a competência, dinamismo e habilidades exigidas pelo mercado de trabalho.

Por sua natureza profissionalizante, o curso destina-se à qualificação profissional ou à capacitação docente para outras Instituições de Ensino, conforme possibilita e determina a Portaria 53/92.

2.1. A QUEM SE DESTINA

Profissionais graduados ou pós-graduados em Análise de Sistemas, Processamento de Dados ou cursos afins, que desejam atualizar e aprofundar seus conhecimentos em Análise de Sistemas; ou ainda profissionais graduados em outras áreas e que exerçam a função de Analista de Sistemas ou funções correlatas.

2.2. REGIME DE ESTUDOS

Os Cursos tem regime semestral, com 15 (quinze) semanas de aula. As disciplinas são oferecidas em 2 (dois) dias da semana. O horário das aulas é das 19:20 às 22:50 horas.

3. OBJETIVO DO CURSO DE ESPECIALIZAÇÃO EM INFORMÁTICA

Com o avanço da Informática nas diversas áreas de atuação, é importante para o profissional, qualquer que seja sua área de atuação, se capacitar com conhecimentos fundamentais na análise e desenvolvimento de sistemas, para poder atuar como projetista de soluções para os problemas de seu dia a dia, senão para melhor interagir como usuário junto aos responsáveis pelo desenvolvimento.

O Curso de especialização em Informática, prevê além dessa capacitação, reciclagem de conhecimentos, objetivando a formação de recursos humanos para a área de Informática, possibilitando a aquisição gradativa de conhecimento.

As disciplinas visam promover a especialização profissional e a capacitação docente, através do embasamento teórico e da abordagem de temáticas atuais, como subsídios para a aplicação prática dos conhecimentos por elas propiciados.

3.1. A QUEM SE DESTINA

Graduados em Curso Superior, que ainda não atuam na área de Informática e que desejam ingressar neste campo de atuação.

3.2. REGIME DE ESTUDOS

Os Cursos têm regime semestral com 15 (quinze) semanas de aula. As disciplinas são oferecidas em 3 (três) dias da semana. O horário das aulas é das 19:20 às 22:50 horas.

Quando do oferecimento da disciplina Linguagem de Programação, além dos 3 dias de aula no horário acima, haverá mais um dia de aula prática no Laboratório de Informática, das 19:20 às 21:00 horas.

4. CRITÉRIOS PARA TITULAÇÃO

4.1. ESPECIALISTA EM ANÁLISE DE SISTEMAS

Para a obtenção do título de ESPECIALISTA EM ANÁLISE DE SISTEMAS, é necessário que o aluno obtenha aprovação em um conjunto mínimo de 6 (seis) disciplinas, totalizando 360 (trezentas e sessenta) horas aula.

Todas as disciplinas são oferecidas na condição de eletivas, exceto para aqueles alunos que desejarem obter capacitação pedagógica, os quais devem, obrigatoriamente, cursar a disciplina de Metodologia de Ensino e Pesquisa em Informática. Para a TITULAÇÃO, os alunos devem ser aprovados em, no mínimo, 360 horas aula.

4.2. ESPECIALISTA EM INFORMÁTICA

Para a obtenção do título de ESPECIALISTA EM INFORMÁTICA, é necessário que o aluno obtenha aprovação nas disciplinas dos grupos de formação básica e de formação complementar e, em algumas de ESPECIALIZAÇÃO, totalizando, no mínimo, 570 (quinhentas e setenta) horas aula, conforme o seguinte quadro:

Grupos de Disciplinas	
Básicas	Obrigatórias
Complementares	Obrigatórias (*)
Especialização	Mínimo de 2 Eletivas

(*) OBSERVAÇÃO: A critério da Coordenação e, eventualmente, através de parecer favorável da Câmara Curricular, o aluno poderá ser dispensado das disciplinas que tiverem sido cursadas em outras Instituições de Ensino Superior ou das quais tenha comprovado conhecimento prático. Porém, a aprovação em, no mínimo, 570 horas aula é requisito indispensável para a TITULAÇÃO.

4.3. EXTENSÃO UNIVERSITÁRIA

Os alunos que forem aprovados em quaisquer das disciplinas isoladamente, como complemento da carga HORÁRIA, ou ainda, na condição de reprovação, têm direito a Certificado de Extensão Universitária nas respectivas disciplinas.

5. RELAÇÃO DE DISCIPLINAS

FORMAÇÃO BÁSICA	HORAS
Introdução à Ciência da Computação	60
Linguagem de Programação	90
Estrutura de Dados e Recuperação da Informação	60

FORMAÇÃO COMPLEMENTAR	HORAS
Tópicos em Linguagem de Programação	60
Análise Estruturada de Sistemas	60
Projeto Estruturado de Sistemas	60
Técnicas e Modelagem de Banco de Dados	60

ESPECIALIZAÇÃO (*)	HORAS
Engenharia de Software	60
Redes de Computadores	60
Inteligência Artificial	60
Arquitetura de Sistemas Computacionais	60
Computação Gráfica	60
Tópicos em Sistemas de Informação I	60
Tópicos em Sistemas de Informação II	60
Tópicos em Gerência de Sistemas I	30
Tópicos em Gerência de Sistemas II	30
Metodologia de Ensino e Pesquisa em Informática	60

(*) O conjunto de disciplinas eletivas poderá sofrer inclusão de novas disciplinas.

A disciplina Metodologia de Ensino e Pesquisa em Informática é obrigatória para os alunos que desejarem obter a capacitação pedagógica para outras Instituições de Ensino.

6. INSCRIÇÃO

A inscrição deve ser feita junto à secretaria de PÓS-GRADUAÇÃO do Instituto de Informática, , Campus I, PUCCAMP, Rod. Dom Pedro I, Km 136, telefone (0192) 52-0899 ramal 195, durante o mês de fevereiro (ou junho).

É necessário o preenchimento de ficha de inscrição acompanhada de:

- Cópia Autenticada do Diploma ou Certificado do Curso Superior,

- Cópia Autenticada do Histórico Escolar do Curso Superior,

- "Curriculum Vitae"

- Cópia Autenticada da Certidão de Nascimento ou Casamento,

- Cópia Autenticada do R.G.

Os candidatos, ainda não inscritos nos Cursos de Especialização, passam por processo de seleção, através da análise da documentação apresentada e, se necessário, de prova escrita e/ou entrevista com a Coordenação do Curso.

Revista do Instituto de Informática
Publicação Semestral do Instituto de Informática
PUCCAMP

A Revista do Instituto de Informática aceita colaborações que lhe forem espontaneamente enviadas, reservando-se o direito de publicá-las ou não, conforme avaliação dos Editores. Os temas abordados serão relacionados com as várias áreas de Informática. Os trabalhos podem ser artigos científicos (textos com 30.000 caracteres), ou opiniões (texto com 8.000 caracteres) e remetidos em 3 vias, seguindo o formato dos artigos e opiniões aqui publicados*.

* Os nomes dos autores, endereço e vinculação profissional devem aparecer em folha separada do texto, de modo a possibilitar, sem identificação, um julgamento do trabalho.