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).

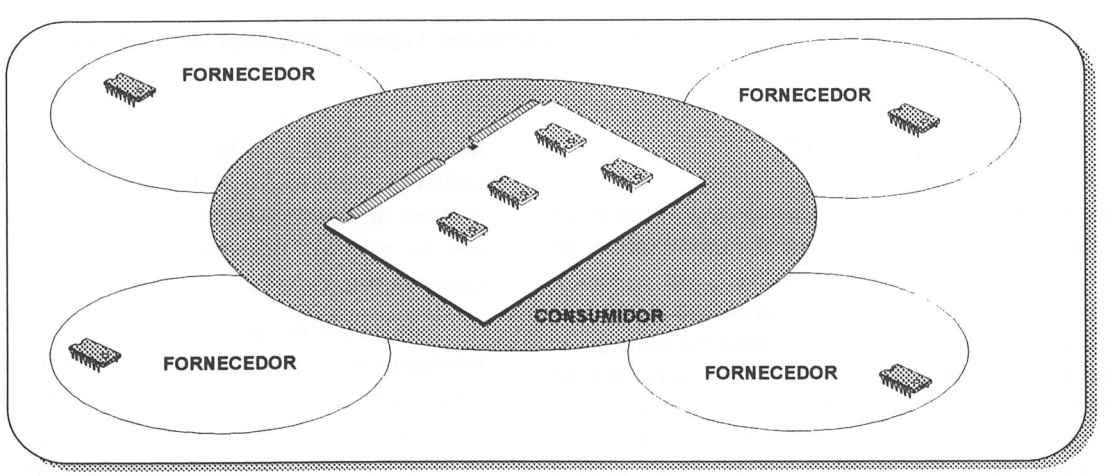

Figura 1 - O usuário pode montar as suas aplicações a partir de um conjunto de componentes adquiridos de diferentes fornecedores.

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.

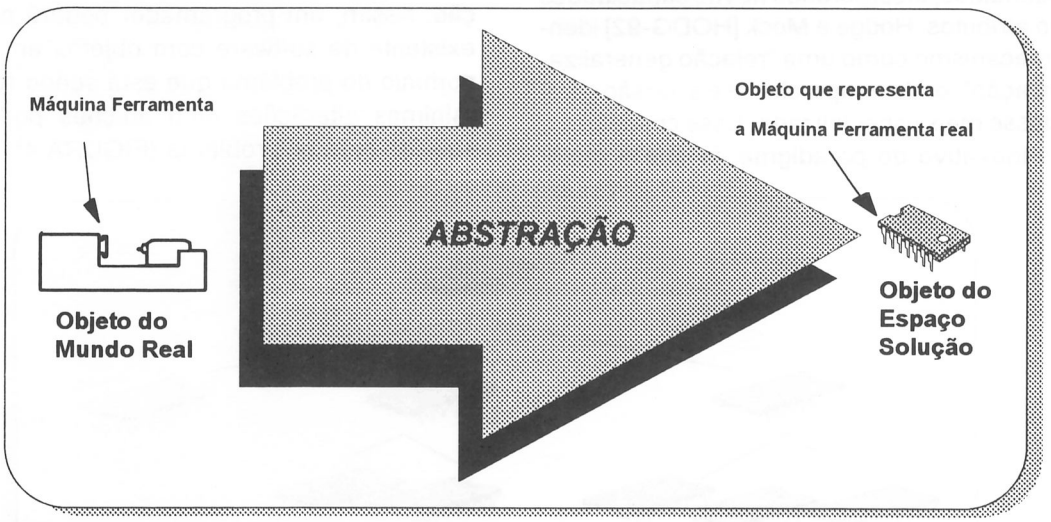

Figura 2 - O conceito de abstração. O programador visualiza uma entidade do mundo real e logo cria uma representação simbólica no espaço solução. Esta representação é chamada de "objeto".

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)

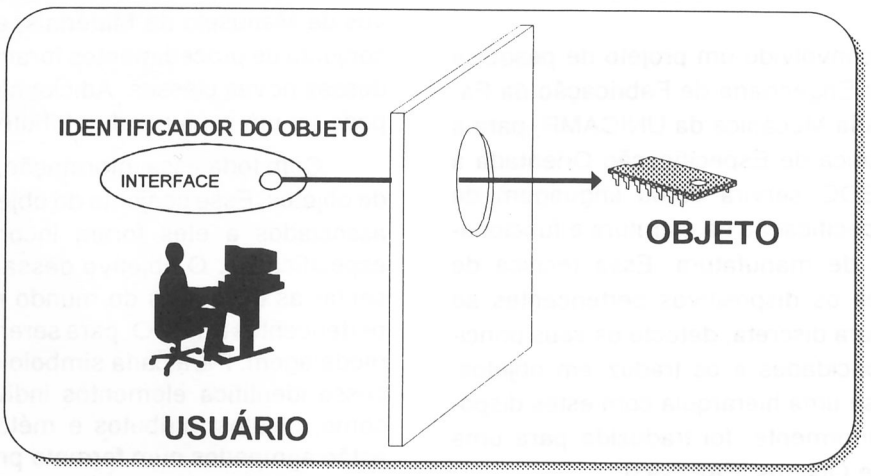

Figura 3 - O Conceito de encapsulamento. Existe uma muralha rodeando cada objeto. O usuário apenas especifica o que ele deseja que o objeto faça através do envio de uma mensagem. A forma como o objeto realiza a ação desejada, ficará oculta numa parte privada do objeto.

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)

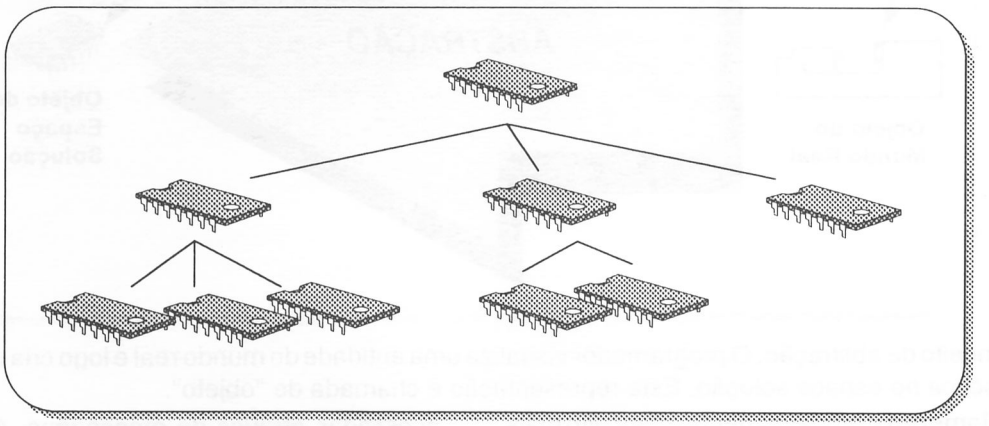

Figura 4 - O conceito de herança. Uma classe primária origina diferentes subclasses. Essas subclasses podem herdar algumas caraterísticas das superclasses e, adicionalmente, incorporar outras caraterísticas próprias.

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.

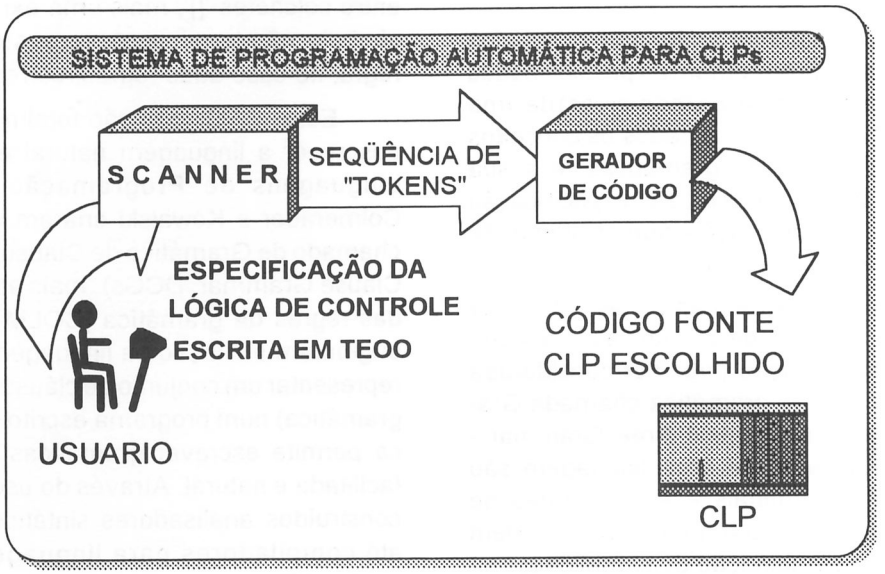

Figura 5 - Estrutura do sistema de geração automática de software para CLPs a partir de uma descrição feita em TEOO.

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.

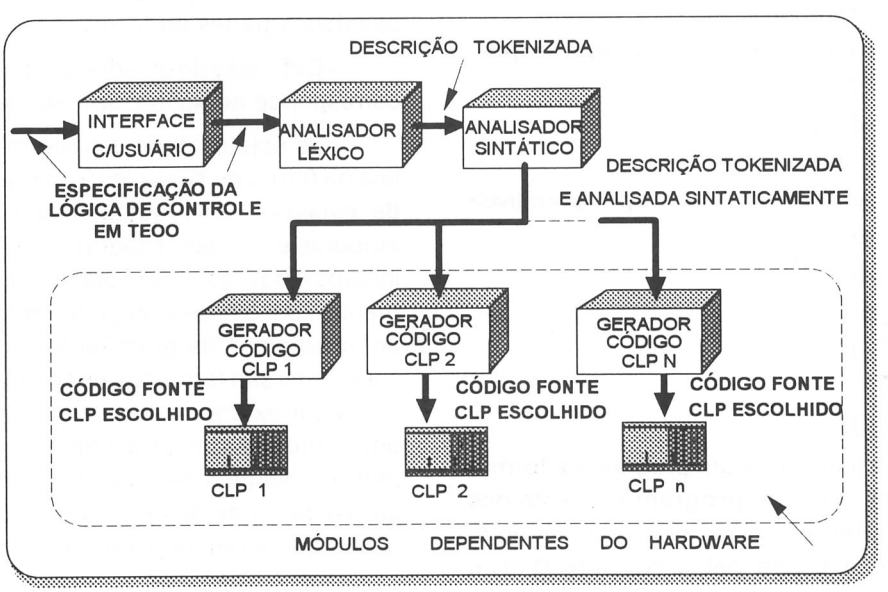

Figura 6 - Módulos do sistema de programação automática para CLPs

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.