Adequação do ambiente A_HAND para o ensino

Alexandre P. Teles, Carlos A. Furuti e Rogério Drummond
Laboratório A_HAND
DCC — IMECC — UNICAMP
13081-970 Campinas, SP
e-mail: teles@dcc.unicamp.br

Junho de 1994

Sumário: O ambiente de programação adotado no ensino de Ciência da Computação influencia tanto o rendimento e produtividade dos alunos como sua competitividade no mercado de trabalho. Além disso, a primeira linguagem de programação utilizada por um aluno tem papel fundamental, pois influencia de maneira marcante, se não permanente, a maneira como ele produzirá programas, desde a abordagem inicial até o produto final. A diversidade de ambientes e linguagens de programação atualmente disponível torna não-trivial e muito controvertida a decisão por um determinado ambiente. Este artigo apresenta os principais tópicos que devem ser considerados nessa decisão e avalia o Ambiente A_HAND quanto à sua adequação como ferramenta de ensino.

Abstract: The programming environment adopted by a Computer Science teaching programme strongly affects students' productivity. Also, the very first programming language usually dictates a student's approach to programming for a long time. The sheer diversity of programming environments and languages available today makes it a difficult decision for the most appropriate combination. Here we present the main issues involved and assess the A_HAND environment as an educational tool.

Palavras-chave: ensino de programação, programação por objetos, ambientes de programação

1. Introdução

A diversidade de ambientes e linguagens de programação existente hoje impossibilita sua cobertura completa em um curso de Ciência de Computação. Cada instituição elege um subconjunto representativo desses ambientes e linguagens para seu programa de ensino.

O ambiente de ensino deve priorizar a assimilação dos conceitos estudados: um aluno que domine bem os conceitos apresentados estará apto a realizar seu trabalho, qualquer que seja o ambiente que efetivamente venha a usar. Por outro lado, o estado e tendências do mercado também devem ser considerados, pois um aluno familiarizado com as linguagens e ferramentas já estabelecidas terá mais chances de sucesso no mercado de trabalho.

Em particular, uma linguagem deve ser eleita para os cursos mais introdutórios do programa. Não há, contudo, um consenso nesse sentido. De fato, um estudo recente (1992) levantou cerca de trezentas escolhas distintas em diferentes instituições.[1]

Algumas linguagens, como Pascal,[11] alcançaram notável prestígio em instituições de ensino. Ainda hoje Pascal é usada em vários cursos introdutórios, porém se apresenta muito restrita em um contexto mais amplo, sendo comumente descartada em cursos mais adiantados como arquitetura de computado-

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

res, sistemas operacionais, redes e compiladores. Uma linguagem que possa dar cobertura a todo o programa de ensino[1] é bastante conveniente, evitando a "latência" (reaprendizado de conceitos já vistos) provocada pela troca de linguagens no decorrer do curso.

Hoje o paradigma de objetos é dominante, e a tendência atual é a migração para linguagens que suportem programação orientada a objetos. Algumas instituições adotaram C++,[2] que apresenta os atrativos de compatibilidade[2] com C[3] (provavelmente a linguagem mais difundida no mercado), além do suporte a objetos. A compatibilidade com C, porém, tem seu lado negativo: forçou uma sintaxe nada agradável para várias das extensões de C++, e herança das idiossincrasias de sua predecessora. Essas características dificultam o aprendizado da linguagem.

1.1. O Projeto A_HAND

O Projeto A_HAND, vinculado ao Departamento de Ciência da Computação (DCC) da Universidade Estadual de Campinas (Unicamp), enfoca principalmente a produção de um ambiente de desenvolvimento de *software* de larga escala. Este ambiente, também denominado A_HAND, é atualmente centrado na linguagem Cm.

A linguagem Cm[7] apresenta recursos comparáveis aos de C++. Ela também foi inspirada em C, porém sem o compromisso de compatibilidade completa. Dessa forma, a sintaxe e semântica de Cm ficou mais simples e concisa. Cm foi projetada enfatizando eficiência de programação, obtida com projeto modular e reaproveitamento de código, com controle automático de dependências entre módulos.

O ambiente A_HAND não é voltado especificamente para o ensino; a ênfase no seu projeto foi a adequação para desenvolvimento de sistemas complexos por grupos de programadores operando em paralelo com razoável independência. No entanto as características do ambiente o tornam perfeitamente viável para uso no ensino.

A seção 2 apresenta de maneira mais ordenada e detalhada os apectos a serem considerados em um ambiente de desenvolvimento de *software*. A seção 3 apresenta Cm em mais detalhes, analisando sua adequação ao ensino e confrontando-a com outras linguagens.

2. Requisitos do ambiente

O ensino de Ciência da Computação pressupõe como ferramentas básicas uma linguagem de programação e uma série de aplicativos de apoio coletivamente conhecidos como ambiente de programação. A linguagem e o ambiente devem satisfazer uma série de requerimentos que, se não exclusivos do contexto de educação, são imediatamente reconhecidos como importantes para a produtividade do aluno e o acompanhamento por parte do instrutor.

2.1. Paradigma de Objetos

Um paradigma de programação é um estilo de criação de programas adequados à resolução de uma certa classe de problemas. Linguagens e ambientes de programação comumente são dotados de características ou restrições favorecendo um ou mais paradigmas em particular. Isso ocorre em diferentes níveis: uma linguagem pode *permitir* o uso de um paradigma embora obrigue o programador a uma série de artifícios e desvios para seguir aquele estilo. Nisso a linguagem difere de outra que *suporta* o paradigma – isto é, encoraja o uso do estilo ou mesmo proíba estilos aberrantes. Suporte a um paradigma não se restringe à linguagem e suas

1. Evidentemente, cursos como linguagem de montagem, inteligência artificial e estudo comparativo de linguagens podem impor o uso de linguagens específicas, fugindo à regra.

2. *backward compatible*

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

construções, abrangendo o ambiente de apoio (ferramentas e sistema de execução).

O paradigma de objetos é o modelo de programação mais requisitado desta década, embora suas origens não sejam recentes e muito de seu potencial seja um refinamento de modelos anteriores como modularidade e abstração de dados. Dada sua importância e influência nas áreas correlatas de análise e projeto, o paradigma de objetos merece suporte ativo num ambiente de ensino de programação. A capacidade de objetos de reduzir o *gap* semântico entre conceitos reais e sua representação computacional auxilia a tarefa do programador iniciante. Por outro lado, objetos vistos como dados ativos comunicando-se via mensagens modelam naturalmente idéias de programação concorrente.

Uma linguagem pode ser puramente orientada a objetos, suportando nenhum outro paradigma; ou pode ser híbrida, apresentando facilidades para manipular objetos enquanto permite ainda programação "convencional". Os argumentos a favor de linguagens híbridas são a familiaridade para programadores de outros paradigmas, e eficiência de execução. Certamente objetos não são necessários para representar *todos* os conceitos de um programa, e o *overhead* acarretado pelo uso de objetos (e.g., passagem de mensagens) pode ser inaceitável em certas aplicações usando a tecnologia atual.

2.2. Segurança

A linguagem e ambiente usados para ensino devem proteger o programador iniciante de seus próprios erros. Isso significa verificar cada possível ação do programa sem restringir significativamente a capacidade do sistema ou seu desempenho.

Tipos são tratados diversamente pelas linguagens de programação. Algumas como Smalltalk não têm tipos ou permitem que objetos mudem de tipo durante sua existência. Linguagens com inferência de tipos podem controlar rigidamente o tipo de um objeto mas dispensam qualquer declaração explícita do programador. Linguagens convencionais usualmente têm tipos estáticos, sendo variável a verificação de compatibilidade oferecida. Uma linguagem para ensino deve incentivar planejamento sistemático, preferivelmente com tipos estáticos e verificação forte de tipos de forma a apontar erros de programação antes da execução de um programa. Se tipos não são estáticos, o sistema de execução é responsável por detectar violações semânticas; essa abordagem implicitamente condiciona a correção do programa aos dados de entrada.

Alguns erros de programação somente podem ser notados durante a execução: violações de endereço, *overflow*, exaustão de recursos, etc. O sistema de execução deve verificar esses problemas; freqüentemente o estado incorreto do programa causado por um erro motiva outro erro e assim sucessivamente ("*Nth-order damage*") – detecção precoce é fundamental para depuração eficiente.

Exceções são uma técnica de programação para melhor isolar o tratamento de casos "excepcionais" do processamento propriamente dito. Consiste em determinar em seções de código bem definidas uma série de condições especiais e as respectivas ações a serem tomadas. A ocorrência de uma dessas condições durante o processamento causa a imediata execução da ação correspondente na seção de exceções mais próxima. Exceções tornam-se assim um mecanismo útil para resolver de forma consistente e uniforme falhas arbitrárias numa cadeia complexa de invocação de subrotinas, principalmente através de módulos diferentes. O programador decide o nível mais apropriado para tratar uma exceção (aquelas não tratadas são propagadas para outros níveis de chamada). Há diferentes estratégias de tratamento, como revogar ou retomar a ação causadora da condição excepcional, usar

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

recursos alternativos, ou recorrer a *graceful degradation*.

Embora poucas linguagens suportem exceções, seus atrativos são claros:

- programas tendem a ser mais simples e fáceis de estender
- código gerado pode ser mais compacto e eficiente para o caso geral, sofrendo penalidades apenas em caso de ocorrência efetiva de exceções
- subrotinas e classes de objetos são potencialmente melhor documentadas incluindo-se a descrição das exceções possíveis e tratadas. Exceções podem ser relacionadas a pré/pós-condições para cada função. Exceções virtualmente *devem* ser documentadas porque, ao contrário de indicadores de erro convencionais, não podem ser ignoradas; além disso, carregam um valor passível de teste durante o tratamento.

Em resumo, embora não substituam completamente construções de decisão convencionais, exceções devem ser seriamente consideradas no projeto de uma linguagem usada em ensino de programação.

2.3. Abstração

Abstração aqui significa qualquer técnica para observar um conceito com reduzido detalhamento de forma a facilitar sua compreensão. Isso se aplica tão bem ao projeto de programas como ao estudo de computação. Nem todos as construções de uma linguagem são essenciais ao aluno iniciante; um rico elenco de construtores de tipo pode a princípio dificultar a escolha da "melhor" alternativa.

O aluno pode observar o ambiente de programação como um núcleo essencial ao qual são progressivamente agregadas facilidades. Cada adição introduz novos conceitos ou detalha idéias apresentadas anteriormente. À medida que o curso progride, diferentes níveis de abstração são revelados, permitindo que o aluno "redescubra" conceitos já dominados adicionando flexibilidade ou eficiência.

Requisitos importantes para abstração na linguagem são uniformidade e ortogonalidade de conceitos (por exemplo, todos os tipos são "cidadãos de primeira classe" e têm as mesmas capacidades).

2.4. Abrangência

Os conceitos aprendidos em qualquer curso introdutório devem continuar válidos quando temas mais avançados são estudados. Sistemas de programação com interesse em educação podem se revelar por demais restritos num contexto mais geral. A linguagem Pascal, explicitamente projetada para ensino quando o paradigma de Programação Estruturada era dominante, é inegavelmente apropriada à descrição clara e elegante de algoritmos convencionais. Sua influência sobre outras linguagens e Computação em geral transcende seus objetivos iniciais. Entretanto, a linguagem é deficiente em encapsulamento de dados e modularização, que são fundamentais ao paradigma de objetos. Por outro lado, a simplicidade de seu mecanismo de entrada/saída e um tratamento excessivamente rígido de tipos, entre outros aspectos, dificultam enormemente a produção de programas genéricos ou não-convencionais. Diversas extensões da linguagem foram propostas para reduzir esses problemas, algumas com notável sucesso comercial mas sempre ao custo da menor portabilidade entre sistemas diversos.

Se possível, as mesmas ferramentas e linguagem devem ser suficientemente abrangentes para serem exploradas de forma continuada durante um programa acadêmico, refinando seu aproveitamento pelo aluno e evitando reaprendizado de temas básicos. Como benefício adicional, projetos de curso podem ser desenvolvidos de forma incremental baseando-se em resultados anteriores.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Um programa acadêmico com uma linguagem de tal flexibilidade poderia usar apenas os conceitos básicos num curso de Introdução à Programação; aspectos de objetos e tipos ao tratar de Estruturas de Dados; abordar encapsulamento e modularidade em Engenharia de *Software*; usar ferramentas específicas na construção de compiladores e outros processadores de linguagem; e caso haja suporte na linguagem, apresentar tópicos de Sistemas Operacionais como mecanismos de concorrência e Programação Distribuída.

Certamente a polivalência de uma linguagem não dispensa o estudo comparativo de outras linguagens, e jamais deve ser conseguida ao custo de excessiva complexidade ou obscuridade.

2.5. Ferramentas de apoio

A única ferramenta de programação essencial é responsável pela execução dos programas; ela toma comumente a forma de um compilador ou um interpretador. Muitas linguagens são implementáveis efetivamente com qualquer das duas abordagens, dependendo a escolha do uso da linguagem. Compiladores são essenciais para a geração de executáveis autônomos. Para a maioria dos casos, também oferecem execução mais eficiente. Uma ferramenta auxiliar ou o próprio compilador é responsável por processar componentes de um projeto com vários módulos. Para propósitos de ensino, interpretadores podem ser mais atrativos. O resultado de alterações no programa é verificável mais rapidamente e o aluno pode normalmente conferir a correção de trechos do programa muito antes dele estar completo.

Um interpretador pode substituir com vantagens muitas funções de um depurador, incentivando a exploração pelo aluno de cenários de execução alternativos. A ferramenta pode ser suficientemente completa para exibir de forma intuitiva o estado do programa, incluindo estruturas de dados complexas. Deve ser contudo enfatizado ao aluno que um ambiente interativo não deve incentivar programação por tentativa-e-erro; tampouco substitui planejamento e especificação cuidadosos do problema.

Efetivamente um ambiente de ensino apropriado também à programação sistemática poderia incorporar as duas técnicas de execução. Seções do programa modificadas interativamente podem ser executadas sem necessidade de compilação, enquanto módulos já completos e testados podem ser compilados para maior velocidade de execução. Mais que um interpretador convencional, a ferramenta incluiria ligação dinâmica de módulos e bibliotecas (dependendo do suporte do sistema de execução) e possivelmente compilação incremental.

Um editor especializado é outra ferramenta importante no ensino de programação. Além de tarefas básicas como busca por componentes em diferentes níveis do programa e consistência automática de delimitadores, o editor pode conhecer a sintaxe da linguagem, virtualmente impossibilitando a inserção de programas com erros léxicos ou sintáticos. Modelos de estilo ou documentação podem ser determinados para o editor e automaticamente inseridos, infundindo padrões ao estudante ao mesmo tempo reduzindo seu esforço. Tal editor seria conectável ao interpretador/depurador formando um ambiente integrado de programação. Estando envolvidos objetos de tipos diferentes num programa com vários módulos, múltiplas janelas de edição são requeridas. Funções de busca e referência facilitam a navegação entre janelas.

A geração de caminhos de teste é tarefa complexa, embora seja importante para o aluno certificar-se que seu programa é completamente exercitado, além de necessário ao acompanhamento do instrutor. Um ambiente com capacidade de *profiling* auxilia o usuário a otimizar seu código enquanto permite uma verificação simples da execução.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Outras ferramentas dependentes da linguagem devem estar disponíveis em cursos mais avançados. Analisadores de código-fonte auxiliam a organização de projetos complexos. Utilitários de geração automática de programas são essenciais ao estudo de implementação de linguagens; o depurador/interpretador pode ser estendido para lidar com objetos invocando métodos concorrentemente ou comunicando-se com outros objetos localizados em diferentes processadores.

2.6. Outros Requisitos

Eficiência

A eficiência de um sistema usado no ensino deve ser medida menos pela otimização dos projetos produzidos que pela velocidade de programação e pela persistência dos conceitos apresentados. Algumas das características favoráveis aqui mencionadas – como abstração e segurança – podem requerer a implementação do ambiente, dos programas produzidos ou ambos de forma não-ótima no tocante ao desempenho. Essa pode ser encarada como uma deficiência menor dados os objetivos do ambiente.

O aluno deve valorizar facilidades de correção, manutenção e reaproveitamento de algoritmos eficientes antes de se preocupar em demasia com o desempenho do código gerado.

Disponibilidade

O ambiente de programação deve estar facilmente disponível ao aluno. Pelo menos suas ferramentas básicas devem ter requisitos reduzidos adaptando-se à grande base instalada de computadores pessoais existentes. Portabilidade entre diferentes ambientes computacionais deve ser enfatizada. Além do ambiente (compilador/interpretador e ferramentas auxiliares), sugere-se que um volume significativo de código-fonte de qualidade (e.g., funções de biblioteca ou um projeto do "mundo real") seja distribuído em conjunto fornecendo ao aluno material de estudo e comparação.

3. A linguagem Cm

Cm é uma linguagem de programação de propósito geral, com facilidades para programação orientada a objetos. Seu objetivo é facilitar a produção de grandes programas pelo projeto modular e reaproveitamento de código. Seus recursos são comparáveis aos de C++, porém sintaticamente mais simples e concisos. Atualmente a linguagem está sendo estendida, incorporando facilidades para programação concorrente e distribuída.

3.1. Visão geral

A linguagem Cm é derivada da linguagem C e em essência suporta um novo tipo: *classe*. Seu objetivo é facilitar a produção de grandes programas pelo projeto modular e reaproveitamento de código. Cm preserva a flexibilidade e a estrutura de comandos de C, adicionando verificação de tipos, encapsulamento de dados, polimorfismo paramétrico e herança múltipla, mantendo ligação estática.

Cm apresenta alta uniformidade de tipos: objetos de qualquer tipo compatível, mesmo estruturado, podem ser comparados, copiados, passados por parâmetro e retornados por funções. Tanto a passagem de parâmetros como o retorno de funções podem ser feitos por referência ou valor. Operadores podem sofrer sobrecarga (*overloading*), tornando ainda maior a uniformidade de tipos.

Expressões e parâmetros de funções sofrem verificação de tipos rigorosa. Enumerados são suportados como subtipos.

Classes introduzem na linguagem suporte à programação modular, abstração de dados e programação orientada a objetos. Cada classe define constantes, tipos, variáveis e funções, públicos ou privados. Cada objeto em Cm com tipo derivado de uma classe contém versões particulares dos dados

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

declarados na classe e usa as funções para prover acesso a elas.

Classes podem ter parâmetros, inclusive expressões de tipo; assim, podem definir um número ilimitado de novos tipos. Uma classe "pilha" pode ser escrita uma única vez, tendo como parâmetros o tipo a armazenar e a capacidade. Qualquer combinação de tipos e constantes inteiras pode ser usada para definir muitos tipos de pilha diferentes. A mesma função para inserir objetos numa pilha é aplicável em diversos pontos do mesmo programa, tendo como parâmetros objetos de tipos diferentes a cada chamada. Esse polimorfismo é comparável ao oferecido pelos *generic packages* em Ada.[4]

Uma classe pode também herdar todas as características de outras, enquanto altera ou adiciona novas propriedades, ou mesmo apenas utilizar outras classes, sem contudo herdar suas características. Programas em Cm constituem uma hierarquia complexa de classes.

Cm apresenta um mecanismo de tratamento de exceções semelhante ao de C++. Exceções podem ser sinalizadas (geradas) e tratadas pelo programador. Podem também carregar informações (um objeto de qualquer tipo), que permitem um tratamento mais seguro e elaborado. Exceções são propagadas automaticamente por vários níveis, e exceções não tratadas provocam a interrupção imediata da execução do programa, sinalizando o evento detectado e o ponto em que ocorreu.

Ainda assim, Cm mantém grande semelhança com a linguagem C, apresentando todos os seus operadores, comandos e tipos padrão (pré-definidos), facilitando a migração de programadores e porte de programas de C para Cm.

3.2. Programação em Cm

Todo programa Cm é por definição uma classe. A execução do programa consiste na criação e destruição de um objeto da classe que o define.[1] Cada classe é definida em um arquivo próprio. Assim, em Cm os conceitos de módulo, classe e arquivo (fonte de programa) se fundem, impondo uma forma mais organizada e modular de programação.

A definição de uma classe se inicia com um preâmbulo, que define o nome, parâmetros de classe, classes herdadas e importadas. Esse preâmbulo, por si só, permite a visualização de todas as relações de dependência da classe.

```
class Complex <type T = double>
import Input, Output;
inherit Cartesian <T>;
```

O preâmbulo das classes é suficiente para que o compilador detecte automaticamente dependências entre arquivos, dispensando o programador de declarações ou ferramentas auxiliares como *make* para controle de compilação. Não sendo necessários conhecimento de detalhes de compilação ou declarações explícitas, evitam-se erros na programação e manutenção, principalmente de sistemas de grande porte.

Todos os elementos (dados, tipos e funções) de uma classe são privados, exceto se exportados explicitamente, passando a fazer parte da *interface* da classe. Não existem elementos globais, no sentido de C. Uma classe somente pode utilizar elementos das interfaces de classes explicitamente importadas ou herdadas. Essas características de Cm reforçam o uso de abstração de dados e, definidas as interfaces das classes, facilitam o trabalho em paralelo de grupos independentes de programadores.

Dessa forma, em Cm o programador é praticamente forçado a um projeto modular,

1. Acarretando a execução de seus construtores e destrutores.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

decompondo seu programa em partes menores (que definem tipos abstratos, ou classes) e estabelecendo a forma como elas se interrelacionam.

Por outro lado, a codificação interna de uma classe é basicamente procedural. Uma classe simples, que implementa pilhas de tipo e capacidade parametrizados, pode ser dada por

```
class Stack <type El; int sz>

El[sz] st;
int top = 0;

export void push (El elem)
{
   st [tóp++] = elem;
}

export El pop ()
{
   return st[--top];
}
```

Com esta classe simples, podemos declarar diferentes objetos pilha:

```
class Test <>
import Stack;

constructor ()
{
   Stack<int,10>              st1;
   Stack<double,8>            st2;
   Stack<Stack<int,10>,3>     st3;

   st1.push (122);
   st2.push (3.1416);
   st3.push (st1);
}
```

3.3. Compilador Cm

A simplicidade de uso do compilador Cm e a semântica da linguagem tornam o uso de Cm fácil e bastante seguro, mesmo para iniciantes. A disponibilidade de fontes do compilador amplia seu aproveitamento em cursos mais específicos como compiladores ou engenharia de *software*, onde a arquitetura e funcionamento do compilador podem ser estudadas.

Simplicidade

O processo de geração do binário executável a partir de um conjunto de fontes Cm é bastante simples, do ponto de vista do usuário. Basta invocar o compilador para a classe principal do programa e os demais arquivos serão buscados, compilados e link-editados automaticamente.

Segurança

O compilador Cm realiza toda a verificação estática imposta pela linguagem em tempo de compilação, como por exemplo verificação forte de tipos na passagem de parâmetros e aplicação de operadores, gerando mensagens de erro sempre que detecta uma violação semântica.

Mensagens de advertência (*warnings*) são geradas para algumas construções que, apesar de válidas, normalmente correspondem a erros ou têm efeitos colaterais supostamente indesejáveis. Algumas dessas construções, porém, são freqüentemente utilizadas por programadores experientes, que aproveitam efeitos colaterais da construção para obter um código mais compacto, e ficariam bastante incomodados com essas mensagens de advertência. Por exemplo, um erro comum a programadores inexperientes é a troca do operador de comparação de igualdade (==) pelo de atribuição (=) em condições (comando if, e.g.); por outro lado, programadores experientes utilizam regularmente construções dessa forma.

Para permitir segurança ao programador inexperiente, mantendo silencioso "respeito" ao experiente, as mensagens de advertência são agrupadas em níveis, que podem ser ativados/desativados seletivamente por chaves de compilação.

Disponibilidade

A versão corrente do compilador Cm, denominada cmc, opera na realidade como um tradutor, produzindo código C padrão (K&R) a partir de arquivos fonte Cm. O compilador C nativo do sistema é então invo-

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

cado de maneira transparente ao programador para geração do arquivo binário executável.

Dessa forma, como o próprio tradutor é também escrito em C e como o código C gerado é portável, Cm é potencialmente disponível em qualquer arquitetura e sistema operacional que possua um compilador C convencional.

Os fontes do tradutor cmc são distribuídos pelo Projeto A_HAND e estão disponíveis via ftp anônimo. A distribuição corrente de Cm já foi utilizada em ambiente Unix (SunOS 4.*, Solaris2.2, Solaris 2.3, SCO/Unix 3.2.4, Linux) e MS-DOS (IBM-PC).

A disponibilidade dos fontes e geração de código em uma linguagem de alto nível torna Cm adequada para projetos em um curso de compiladores. Tendo os fontes à mão, o aluno pode analisar o funcionamento do compilador e realizar extensões à linguagem, ou mesmo reimplementar partes do compilador previamente extraídas pelo instrutor. O uso de C como linguagem alvo permite a abstração de detalhes de máquina, facilitando a geração de código, além de possibilitar uma cobertura mais ampla dos principais conceitos em um curso de compiladores.

3.4. Sistema de execução (*run-time*)

Uma vez que programas Cm são traduzidos para C, o sistema de execução usado é praticamente o mesmo de C, estendido para prover a funcionalidade necessária a Cm.

O *run-time* específico da linguagem é bastante simples, praticamente voltado para gerenciamento de memória e controle de exceções. Uma versão mais elaborada desse *run-time* com facilidades para *tracing* e *profiling* está sendo projetada.

O mecanismo de controle de exceções impede a continuidade da execução do programa quando uma exceção (erro) ocorre e não é tratada. Nesse caso é gerada uma mensagem (*run-time error*) explicitando o ponto (nome do arquivo fonte e número da linha) e natureza (tipo e valor associado à exceção) do erro ocorrido, impedindo que ele passe despercebido.

Por outro lado, o sistema de execução não provê suporte (além do provido pelo sistema de execução de C) para detecção de problemas como acesso ilegal à memória, liberação dinâmica (delete/free) de memória alocada estaticamente, uso de memória não alocada ou já liberada, etc. Algumas verificações desse porte são computacionalmente dispendiosas mas aceitáveis em casos específicos, como no ensino da linguagem ou na depuração de programas. A flexibilidade de Cm permite a solução parcial deste problema, pela definição de tipos de dados (através de classes) com manipulação controlada e funcionalidade próxima à dos originais.

3.5. Biblioteca padrão

A atual biblioteca padrão de Cm apresenta classes para entrada e saída, acesso ao ambiente (argumentos de linha de comando, variáveis de ambiente), interface com o usuário[13] (gráfica e modo texto), acesso (parcial) à biblioteca padrão de C e estruturas de dados simples (*strings*, pilhas, listas, *arrays*, etc). Qualquer biblioteca C, em particular a biblioteca padrão (libc), pode ser utilizada através de declarações externas.

Algumas características de Cm (e de C e C++), como o tratamento de *arrays* e apontadores, geralmente apresentam uma fonte de problemas para programadores inexperientes. Classes projetadas especificamente para o ensino podem remover grande parte desses problemas potenciais, sem no entanto prejudicar a qualidade/abrangência do aprendizado ou modificar o estilo de programação. Por exemplo, os mecanismos de Cm permitem a implementação de uma classe para *arrays* com verificação de índices, como os de Pascal, com utilização

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

praticamente idêntica à dos *arrays* originais, e ainda com tratamento mais uniforme em passagem de parâmetros e retorno de funções.

O acesso direto a bibliotecas externas (escritas em C) é problemático com relação a exceções. Uma vez que C não apresenta tratamento de exceções, ocorrências de erros na biblioteca são indicadas por artifícios como valores especiais no retorno de funções, funções de *callback*, variáveis globais ou uma mescla desses mecanismos. Essa diversidade é inconveniente mesmo para programadores experientes, e pode ser evitada provendo-se, na biblioteca padrão Cm, um conjunto pré-definido de exceções padrão e classes de interfaceamento com as bibliotecas C, que convertam indicações de erro em exceções padrão.

Ainda, bibliotecas de classes podem ser escritas em função de cursos particulares, oferecendo subsídios adequados a projetos para melhor entendimento dos conceitos estudados, e reduzindo a preocupação com detalhes externos aos interesses do curso. Por exemplo, para cursos de processamento de imagens poderiam ser implementadas classes padrão para representação e visualização de imagens ou leitura e escrita de imagens em arquivos, permitindo que o aluno se dedique aos aspectos que realmente interessam ao curso, como implementação de filtros de manipulação de imagens.

3.6. Ferramentas de apoio

O Ambiente A_HAND provê diversas ferramentas de apoio ao desenvolvimento. Outras ferramentas serão ainda implementadas ou adaptadas para uso confortável e seguro com Cm.

edição e compilação

A edição de programas Cm é bastante facilitada por um modo de edição para o editor GNU-Emacs. Neste modo o editor "conhece" parcialmente a gramática da linguagem, auxiliando na formatação do texto e identificando visualmente categorias diferentes de elementos como literais, palavras reservadas e comentários.

Uma ferramenta similar ao *ctags* (para C) permite a criação de arquivos com tabelas de referências cruzadas, que pode ser utilizado pelo editor na busca da definição de um determinado identificador, por exemplo.

Certamente o editor é integrado ao compilador (e futuramente ao depurador) simplificando a invocação do compilador e análise de eventuais erros.

Análise estática de código-fonte

O compilador Cm permite a geração de listagens anotadas do programa, gerenciando automaticamente herança e importação de classes.

Outras ferramentas (incorporadas ao próprio compilador, em alguns casos) estão previstas, permitindo gerar

- listagens do programa, opcionalmente com formatação e referências cruzadas de identificadores, classes e funções;
- a interface completa de uma classe, incluindo identificadores herdados (em qualquer nível) e indicando as seleções de escopo mínimas para acesso sem ambigüidade a cada um deles;
- levantamento dos pontos em que uma função é utilizada, considerando o mecanismo de sobrecarga e herança, e a utilização implícita de funções (no caso de construtores, destrutores e definições de operadores);
- resultados do mecanismo de resolução de sobrecarga, para cada função ativada (explícita ou implicitamente).

Análise em tempo de execução

As principais ferramentas utilizadas para análise dinâmica do código são depuradores, interpretadores e *profilers*. Além dessas fer-

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

ramentas, algum suporte pode ser provido pelo sistema de execução.

O sistema de execução de Cm permite acompanhar o funcionamento do mecanismo de tratamento de exceções, indicando o estabelecimento e desativação de tratadores e a sinalização, propagação e tratamento de exceções. Dessa forma, tanto a análise do comportamento de um programa diante de situações excepcionais como o entendimento deste mecanismo é bastante facilitado.

As ferramentas disponíveis para C (depuradores, *profilers*, etc) podem ser usadas sobre o código C gerado, porém seu uso somente será efetivo com um certo conhecimento dos mecanismos de tradução de código do compilador.

O compilador Cm será adaptado ainda este ano para geração de código em RTL e utilização do *backend* da Gnu Software, para geração do código binário.[12] Este enfoque manterá a disponibilidade de Cm (em arquiteturas suportadas pelo gerador de código da GNU), e permitirá a utilização mais efetiva das ferramentas de depuração e *profiling* disponíveis atualmente.

Prevê-se a implementação de um interpretador de Cm, provendo também facilidades para execução de programas parcialmente escritos. Esse interpretador pode ser particularmente útil no ensino, facilitando a experimentação por parte dos alunos.

Geradores de código

Ferramentas para geração automática de código são bastante difundidas hoje em dia. Algumas delas se tornaram indispensáveis em ambiente Unix, como geradores de *parsers* (*yacc*[14] e *lex*[15]). O ambiente dispõe atualmente de ferramentas para a geração de

- analisadores léxicos (porte do GNU-*flex*);
- parsers (porte do GNU-*bison*);
- linearização de estruturas de dados complexas[17] (uma extensão do XDR, comum em Unix).

Estão sendo implementados um gerador de interfaces com usuário (para terminais gráficos e baseados em texto) e uma ferramenta para auxílio na tradução de arquivos C para Cm[16] – a tradução totalmente automática é possível mas nem sempre conveniente para um melhor aproveitamento das vantagens de Cm; por exemplo, a criação de classes representativas dos conceitos envolvidos no programa C permitem obter um código Cm de melhor qualidade, além de tornar estes conceitos disponíveis para utilização em outros programas.

3.7. Cm distribuído

Sistemas distribuídos estão cada vez mais presentes em Ciência da Computação. A disseminação de redes locais estimula a construção de aplicações que se beneficiam da flexibilidade, extensibilidade, tolerância a falhas e relação custo/benefício desses ambientes. Extensões à linguagem Cm foram projetadas para facilitar seu uso no desenvolvimento de aplicações distribuídas bem estruturadas, robustas e eficientes; elas permitirão a criação de objetos distribuídos, controle de concorrência, e comunicação transparente ou propagação de exceções inter–processos.[8]

Objetos remotos em Cm

A programação distribuída em Cm está baseada no conceito de *objetos remotos*. Cada objeto remoto criado constitui um fluxo de execução (*thread*) independente, possivelmente em uma máquina distinta da de seu criador; todos os *threads* podem estar simultaneamente em execução.

Concorrência

Objetos remotos podem implementar servidores, atendendo simultaneamente a pedidos de múltiplos clientes. A sincronização é obtida através de regiões críticas condicionais,

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

baseadas na proposta de Hoare[9]: nelas os recursos compartilhados são referenciados de forma segura.

A forma de acesso a um objeto remoto é idêntica àquela de objetos "comuns": através de seleção de elementos de sua interface. Chamadas a métodos e acessos a dados de objetos remotos são mapeadas em RPCs (*Remote Procedure Calls*); parâmetros são passados e resultados recebidos de maneira transparente ao programador. Exceções propagadas pelo objeto remoto na execução de um método são passadas transparentemente para o objeto que realizou a chamada.

Dessa forma a programação concorrente e distribuída em Cm é bastante semelhante à "convencional", permitindo o desenvolvimento de sistemas distribuídos de maneira simples e sistemática, mesmo por programadores com pouca experiência prévia na área.

Esse mecanismo, integrado à linguagem e seu sistema de execução, permite uma flexibilidade e consistência dificilmente alcançadas por ambientes que implementam tais facilidades por bibliotecas ou mecanismos externos à linguagem.

3.8. LegoShell

A LegoShell consiste de uma *shell* com interface gráfica para manipulação de objetos distribuídos. Através dela um sistema distribuído pode ser configurado compondo-se diversos objetos e se estabelecendo a forma de comunicação e as máquinas onde serão executados. Grupos de objetos relacionados são agrupáveis num único objeto, permitindo a visualização do sistema em diferentes níveis de abstração. Uma vez configurado, um sistema pode ser executado e monitorado através da *shell*. O resultado pode ser um programa Cm autônomo, executável independentemente da *shell*.

4. Outras linguagens

Outras linguagens de programação oferecendo atrativos no tocante à abstração de dados, suporte a objetos e abrangência incluem C++, Eiffel, extensões de Pascal e Modula-3.

A linguagem C++ é funcionalmente equivalente a Cm (o programador proficiente em Cm não terá dificuldades em se adptar a C++). Apresenta como desvantagens sintaxe mais complexa e obscura, gerência de código–fonte inerentemente mais intrincada e ausência de suporte a objetos distribuídos e concorrência (prevê-se suporte por bibliotecas, mas não na própria linguagem, abrindo margem a incompatibilidades por dependência do sistema de execução). C++ dispõe de sólida base instalada e deve substituir C como linguagem de programação de sistemas.

Diversas implementações de Pascal provêem extensões voltadas principalmente à modularidade e objetos, freqüentemente com ambientes de programação integrados. Compatibilidade entre extensões proprietárias e disponibilidade são aspectos negativos.

Implementações de Smalltalk[5] oferecem um ambiente completo com interpretador, editor, inspetor de classes e uma extensiva coleção de classes razoavelmente padronizadas. A linguagem favorece uma abordagem interativa e exploratória; além disso é consistente, levando o conceito de objetos a todos os tipos. Apresenta como desvantagens proteção de acesso a dados reduzida ou inexistente, e ausência de verificação estática do programa.

Modula-3[10] oferece suporte para objetos, exceções, *threads* e aspectos positivos de segurança (*garbage collection*, seções inseguras explicitamente demarcadas). Modula-3 não permite sobrecarga de operadores. A linguagem é bastante concisa e uniforme, e

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

parece promissora. Apesar disso, ainda é pouco difundida.

Eiffel[6] apresenta aspectos originais e interessantes, como a definição de invariantes de classe e pré e pós-condições (para execução de métodos) integradas ao mecanismos de herança, permitindo a reusabilidade de classes e redefinição de métodos de maneira mais controlada. Eiffel também é promissora, porém se trata de um produto comercial, restringindo sua disponibilidade.

5. Conclusões

Cm é uma linguagem complexa e seu domínio completo dificilmente será alcançado em um curso introdutório de Ciência da Computação, pois exige conceitos não dominados pelos alunos. Uma análise superficial, enfocando somente as linguagens de programação, pode considerar mais adequado o uso de uma linguagem mais simples e restrita como Pascal.

Porém, uma análise mais ampla, envolvendo o ambiente (linguagem, compilador, sistema de execução, bibliotecas e ferramentas de apoio) e o mercado atual de informática, indica que o uso de uma linguagem mais elaborada como Cm é mais adequado e vantajoso.

O ambiente atualmente disponível ainda pode ser melhorado em diversos aspectos, mas já pode ser usado com certo conforto no ensino, bastanto a implementação, se for o caso, de bibliotecas de classe Cm abstraindo detalhes de menor importância ou ainda não dominados pelo aluno, para que este se concentre no que realmente deve aprender.

6. Bibliografia

[1] *C++ as a first programming language*, C++ Report 5 (4), May 1993.

[2] *The C++ Programming Language* (2nd ed.), Bjarne Stroustrup, Addison-Wesley Publ. Co., 1991.

[3] *The C Programming Language* (2nd ed.), Brian W. Kernighan e Dennis M. Ritchie, Prentice–Hall Inc., 1988.

[4] *Rationale for the design of the Ada programming language*, J. D. Ichbiah, J. G. Jones, J. C. Heliard, B. Krieg-Bruckner, O. Roubine, B. A. Wichmann, ACM Sigplan Notices 14 (6), junho 1979.

[5] *Smalltalk–80 The Language and its Implementation*, A. Goldberg, Addison–Wesley, 1983.

[6] *Eiffel, the Language*, B. Meyer, Prentice–Hall, NJ, 1992.

[7] *A linguagem de programação Cm*, A. P. Teles, Tese de Mestrado, DCC-UNICAMP, 1993.

[8] *Objetos distribuídos em Cm*, R. Drummond e C. Gonçalves Jr., Anais do XII Simpósio Brasileiro de Redes de Computadores, Curitiba, 1994.

[9] *Towards a Theory of Parallel Programming*, C. A. Hoare, *Operating Systems Techniques*, Academic Press, NY, 1972

[10] *Modula-3 Report*, L. Cardelli, J. Donahue, L. Glassman, M. Jordan, B. Kalsow, G. Nelson, DEC Systems Research Center, 1988.

[11] *Pascal User Manual and Report*, K. Jensen e N. Wirth, Springer–Verlag, NY, 1975.

[12] *Using and Porting GNU CC*, Richard Stallman, Free Software Foundation, 1991.

[13] *The Astra User Interface Library*, C. A. Furuti, VII Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, 1993.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

[14] *Yacc: yet another compiler-compiler*, S. C. Johnson, Relatório técnico, Bell Laboratories, 1978.

[15] *LEX - Lexical Analyser Generator*, M. E. Lesk and E. Schimdt.

[16] *Suporte à Programação em Cm*, Maria Claudia Borges Barros, Relatório técnico, Projeto A_HAND, 1992.

[17] *Linear – linearizador de estruturas complexas*, Rogério Drummond e Marcelo Augusto Holloway de Souza, Relatório técnico, Projeto A_HAND, 1992.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor