Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Linguagens de Programação

Notas de aula para a disciplina EA877

Mini e Microcomputadores: Software

Ricardo R. Gudwin

Departamento de Engenharia de Computação e Automação Industrial

1997

Índice

1. Introdução

Para se implementar um algoritmo em um computador, é necessário descrevê-lo de uma forma que o computador esteja apto a executá-lo. Essa descrição é feita por intermédio de uma "linguagem de programação". O próprio conjunto de instruções de um processador pode ser entendido como uma "linguagem de programação" . Entretanto, essa linguagem normalmente não é a mais adequada para a descrição de um programa, uma vez que os algoritmos necessários podem ser sofisticados, e essa linguagem primitiva, também chamada de "linguagem de máquina" não é nem um pouco amigável ao programador, demandando um esforço muito grande na elaboração de programas mais complexos. Sendo assim, foram desenvolvidas, ao longo da história da computação, diversas "linguagens de programação", cada qual, a seu tempo, introduzindo facilidades e recursos que foram tornando a tarefa de programar mais fácil e menos suceptível a erros. Atualmente, com as linguagens visuais (também chamadas por alguns de linguagens de quarta geração), programar deixou de ser uma arte restrita a um grupo de indivíduos, para tornar-se uma ferramenta a mais dentro do escopo do usuário comum.

Neste texto, apresentamos um tutorial histórico sobre as linguagens de programação, ressaltando aquelas que foram mais proeminentes dentro de cada categoria. As linguagens de programação podem ser divididas dentro da seguinte taxonomia:

- Linguagens de Baixo Nível
- Linguagens Não-Estruturadas
- Linguagens Procedurais
- Linguagens Funcionais
- Linguagens Orientadas a Objeto
- Linguagens Específicas a Aplicações
- Linguagens Visuais

2. Linguagens de Baixo Nível

Linguagens de baixo nível são linguagens cujas instruções correspondem quase que diretamente ao código de máquina que será enviado ao processador para execução. Na verdade, existem tantas linguagens de baixo nível quantos são os conjuntos de instruções dos diferentes processadores. Essas linguagens são conhecidas de uma maneira unificada como "Linguagem Assembly", sendo que na verdade deve existir uma linguagem Assembly para cada processador. Sendo assim, deve haver um Assembly 8086, um Assembly 68000, um Assembly 80386, um Assembly Pentium e assim por diante.

2.1 Assembly

A linguagem Assembly corresponde a uma linguagem em que cada instrução de um determinado processador é associada a um termo, cuja semântica corresponde à operação efetuada pela instrução. Sendo assim, para cada processador, pode existir pelo menos uma linguagem Assembly correspondente. Para um mesmo processador, podem existir também mais de uma linguagem Assembly. Uma linguagem Assembly mais simples, vai apenas nomear as

instruções do processador, substituindo-se as instruções numéricas por termos linguísticos. Linguagens Assembly mais sofisticadas permitirão também a associação de termos às posições de memória apontadas pelas instruções, dando maior flexibilidade ao programador. Entretanto, apesar de serem mais flexíveis, estas linguagens estão ainda bem longe da flexibilidade permitida pelas linguagens de alto nível.

3. Linguagens Não-Estruturadas

Linguagens não-estruturadas são linguagens mais sofisticadas que as linguagens de baixo nível. São linguagens mais flexíveis, pois seus comandos não estão tão vinculados ao processador e sistema utilizados, tornando seu uso possível em diferentes plataformas. Do mesmo modo, a semântica de seus termos torna-se mais genérica, estando desassociada do conjunto de instruções que irão efetivamente implementá-la durante a execução do programa. Isso permite que operações mais sofisticadas sejam emuladas por sequências de instruções mais simples em linguagem de máquina. As linguagens não estruturadas corresponderam a um grande salto qualitativo em termos de programação, quando comparadas com as linguagens de baixo nível. Entretanto, tornaram-se gradualmente obsoletas, com a introdução de linguagens estruturadas mais sofisticadas, tais como as linguagens procedurais, as linguagens funcionais e as linguagens orientadas a objetos.

3.1 Cobol

O COBOL (COmmon Business Oriented Language), foi designado especificamente para a construção de aplicações comerciais, tais como folhas de pagamento, inventário de produtos, contabilidade, operando tipicamente em grandes volumes de dados. Sua estrutura não é adequada, portanto, para problemas científicos, onde cálculos complexos são necessários. Encontra-se disponível em várias plataformas, desde máquinas PC até mainframes. O mesmo programa COBOL pode ser compilado e executar em uma grande variedade de máquinas, com apenas pequenas mudanças no código.

A linguagem COBOL foi desenvolvida em 1959 por um grupo chamado Comitê CODASYL (Conference on Data Systems Languages), que inclui representantes de origem acadêmica, bem como fabricantes de computadores. O objetivo desse grupo foi o de desenvolver uma linguagem padrão para o desenvolvimento de programas comerciais, de tal forma que todos os fabricantes pudessem oferecer compiladores para tal linguagem. A conferência foi patrocinada pelo Departamento de Defesa dos Estados Unidos (DOD). Como resultado do CODASYL, o primeiro compilador COBOL veio ao mercado em 1960. Entretanto, devido à diversidade entre os compiladores COBOL que surgiram, decidiu-se elaborar uma norma para regulamentar a linguagem. Em 1968, a primeira versão do ANSI-COBOL foi desenvolvida e aprovada. A maioria dos grandes fabricantes de computadores e distribuidores de software começaram a distribuir compiladores utilizando o ANSI-COBOL. Em 1974, uma segunda versão foi elaborada e em 1985, uma terceira, conhecida como COBOL-85. Atualmente, esforços vêm sendo empregados na elaboração de uma nova revisão na linguagem, com vistas à uma futura versão, possivelmente a ser anunciada em 1997 (COBOL-97). Algumas das características anunciadas para o novo COBOL incluem orientação a objeto e programação visual.

3.2 Basic

A linguagem BASIC (Beginners All-purpose Symbolic Instruction Code) foi desenvolvida por Thomas E. Kurtz e John G. Kemeny, membros do departamento de matemática de Dartmouth, em 1963. O BASIC foi desenvolvido com o intuito de ser uma linguagem fácil e interativa, de modo a possibilitar seu uso por cientistas da computação, que rapidamente pudessem criar programas e executá-los. Ela se tornou popular nos primeiros microcomputadores. Uma de suas características mais marcantes é o fato de ser uma linguagem interativa, ou seja, dá facilidades ao programador para ir programando ao mesmo tempo em que executa. É uma linguagem frequentemente interpretada, havendo entretanto, compiladores para ela. Não é uma linguagem padrão entre computadores, apesar de existir uma norma ANSI que define a linguagem, existindo mais de 80 diferentes variantes, com diferenças às vezes significativas entre elas. Algumas implementações mais recentes incorporam recursos de programação visual e orientação a objeto (Visual Basic).

4. Linguagens Procedurais

As linguagens procedurais são um dos sub-tipos das linguagens chamadas "estruturadas". Essas linguagens se diferenciam das linguagens não estruturadas, pela existência de estruturas de controle, que entre outras coisas promovem o teste de condições (if-then-else), controlam a repetição de blocos de código (for, while, do), fazem a seleção de alternativas (switch, case), e dividem o código do programa em módulos chamados de funções ou procedimentos. As linguagens procedurais são caracterizadas pela existência de algoritmos, que determinam uma sequência de chamadas de procedimentos, que constituem o programa. Essa característica é colocada em contraposição às linguagens funcionais, onde se descrevem expressões que caracterizam um certo tipo de conhecimento.

As linguagens procedurais mais comuns são o C, o Pascal e o Fortran. Algumas linguagens mais sofisticadas, tais como o Ada, Modula-2 e Modula-3 foram desenvolvidas para corrigir deficiências e explorar novos contextos onde o C, Pascal e Fortran não eram totalmente adequados.

4.1 C

O desenvolvimento da linguagem C está intimamente conectado ao desenvolvimento do sistema operacional Unix. Durante o desenvolvimento do Unix, foi necessária a criação de uma linguagem de alto nível para a programação do sistema, uma vez que o esforço de programação em linguagem assembly era muito grande. Sendo assim, um grupo de pesquisadores do Bell Labs, incluindo Ken Thompson e Dennis Ritchie, desenvolveram uma linguagem de alto nível a qual batizaram de "linguagem B". A linguagem B foi posteriormente substituída por outra que a sucedeu, a "linguagem C". Com o desenvolvimento de um compilador C portátil (por Steve Johnson, do Bell Labs), o Unix pôde ser transportado facilmente para diversas plataformas, tornando tanto a linguagem C como o sistema operacional Unix um sucesso para diferentes plataformas computacionais.

O grande sucesso da linguagem C está intimamente ligado ao sucesso do sistema operacional Unix, sendo que quase todas as implementações do Unix (para diferentes arquiteturas) possuem um compilador C nativo. Posteriormente, devido a

esse sucesso, a linguagem C passou a ser utilizada também fora do ambiente Unix, como por exemplo em PC´s (DOS, Windows e OS/2), com a introdução dos compiladores da Borland e Microsoft.

Atualmente, a linguagem C é descrita pela norma ISO - ISO/IEC 9899:1990.

4.2 Pascal

O Pascal é uma linguagem que foi originada no Algol, tendo sido desenvolvida por Niklaus Wirth para o CDC 6600, por volta de 1967-68, como uma ferramenta de instrução para programação elementar. Apesar de ter sido desenvolvida inicialmente apenas com propósitos educacionais, tornou-se posteriormente uma linguagem de propósitos gerais, tendo-se tornado inclusive o ancestral de diversas outras linguagens, tais como o Modula-2 e o Ada. É uma linguagem que enfatiza bastante sua sintaxe, evitando artifícios encontrados em outras linguagens, tais como "casts", para efetuar conversão de tipos, ou outros artifícios que poderiam tornar a linguagem mais flexível. Neste sentido, o Pascal foi perdendo lugar para o C, que tem uma estrutura semelhante, e ao mesmo tempo dá ao programador a flexibilidade necessária para resolver problemas onde a sintaxe rígida do Pascal não permite muita elaboração.

4.3 Fortran

O "IBM Mathematical FORmula TRANslating system", ou FORTRAN, foi a primeira linguagem de alto nível (ou seja, superior ao assembly). Desenvolvida por Backus, dentre outros, a linguagem foi introduzida em 1958, direcionada primeiramente para cálculos científicos, de tal forma que facilidades como manipulações de "strings" eram quase inexistentes, e a única estrutura de dados existente era o "array". Entretanto, tratou-se de um grande salto qualitativo, quando comparado ao Assembly. Tornou-se uma linguagem de grande aceitação, sendo utilizada até hoje em aplicações numéricas.

A primeira norma regulamentando a linguagem veio entretanto somente em 1966, quando a "American National Standards Institute" (ANSI) publicou uma norma que unificou a linguagem, sendo referenciada como FORTRAN´66. Depois disso, a linguagem passou por diversas revisões, de modo a incorporar idéias da programação estruturada. As mais notáveis revisões, foram o FORTRAN´77 e o FORTRAN´90, introduzidas em 1978 e 1991 , respectivamente. O FORTRAN´90 incorpora diversas características das linguagens modernas de programação, dentre estas o uso de "arrays" dinâmicos, "records", módulos e apontadores, derivação de tipos e "overload" de operadores, um melhor sistema de declarações e o uso de protótipos e modernas estruturas de controle (SELECT CASE, EXIT, ...).

O FORTRAN´90 é descrito pela norma internacional ISO/IEC 1539:1991 (E). O comitê X3J3 da ISO, que é dedicado exclusivamente ao desenvolvimento do FORTRAN, está, no momento, preparando uma nova versão da linguagem, que deverá incluir, dentre outros, suporte para processamento paralelo e programação orientada a objetos.

4.4 Ada

Ada é uma linguagem de programação moderna e avançada, planejada e padronizada de modo a encorajar alguns princípios fundamentais da engenharia de software: confiabilidade, portabilidade, modularidade, reusabilidade, eficiência,

facilidade de manutenção, ocultamento de informação, tipos abstratos de dados, generalidade, programação concorrente e mais recentemente programação orientada a objetos.

O Ada foi introduzido em 1978, desenvolvida por um grupo de projeto internacional, como resposta aos requisitos levantados pelo Departamento de Defesa dos Estados Unidos. Seu desenvolvimento foi inspirado originalmente pelo Pascal e pelo Fortran 77. Em 1983, a linguagem foi transformada em uma norma, a ANSI/MIL-STD-1815A. Em 1995, uma nova norma, a ISO/IEC 8652:1995(E) introduziu o chamado Ada 95, onde uma série de inovações foram incorporadas. Dentre outras, o Ada 95 inclui suporte para programação orientada a objetos, sincronização orientada a dados, programação in-the-large e sistemas de tempo real.

Uma das principais vantagens do Ada em relação às outras linguagens procedurais mais usuais é seu suporte para programação em tempo real, o que inclui primitivas para sincronização de processos e programação concorrente.

4.5 Modula-2

O Modula-2 é uma linguagem de programação desenvolvida por Niklaus Wirth (o mesmo que desenvolveu o Pascal), no final dos anos 70. É uma das linguagens descendentes do Pascal, assim como o Oberon e o Modula-3. Suas principais características são a existência de um rígido sistema de tipos, módulos, uma rica variedade de tipos diferentes, um poderoso conjunto de estruturas de controle, tipos de procedimentos, tipos opacos e co-rotinas.

Até a pouco tempo atrás, não existia uma norma formal para a linguagem Modula-2. O texto clássico descrevendo a linguagem pode ser encontrado no livro “Programming in Modula-2” 3a. edição, publicado pela Springer-Verlag em 1985. Um comitê da ISO, o ISO/JTC1/SC22/WG13 vem, desde 1987, estendendo a linguagem a cada ano. O resultado dos esforços desse comitê resultou na norma ISO IS10514, que foi votada recentemente e é agora a norma oficial para a linguagem. O WG13 estuda no momento duas normas adicionais, uma que define o Modula-2 orientado a objetos e outra que determina facilidades genéricas para a linguagem.

4.6 Modula-3

O Modula-3 é outra linguagem descendente do Pascal e do Modula-2, desenvolvida no final dos anos 80, pela Digital Equipment Corporation e a Olivetti. O Modula-3 corrige diversas deficiências do Pascal e do Modula-2, no que concerne às práticas modernas de engenharia de software. Em particular, o Modula-3 mantém a simplicidade e segurança no uso dos tipos, ao mesmo tempo que provê facilidades no manuseio de exceções, concorrência, programação orientada a objetos e “garbage collection” automático. O Modula-3 é ao mesmo tempo uma linguagem prática para a implementação de projetos de software e uma excelente linguagem para o ensino de linguagens de programação.

Assim como o C, o Modula-3 permite que o programador tenha acesso direto ao “hardware”, por meio de instruções especiais (programação em baixo nível), ao mesmo tempo em que tem mecanismos que protegem o acesso a esses recursos (tais como memória, acesso a discos, etc), durante as instruções “normais”. A implementação SRC do Modula-3 conta com diversos recursos modernos, tais como o uso de “threads” e “streams” . A implementação SRC conta

ainda com um completo conjunto de bibliotecas para os mais diversos usos, tais como funções de GUI e construção de aplicações gráficas. Conta ainda, com a definição de "objetos de rede", que permitem a criação de programas concorrentes com grandes vantagens sobre os esquemas tradicionais de programação distribuída.

5. Linguagens Funcionais

Linguagens funcionais são linguagens que evidenciam um estilo de programação diferente das linguagens procedurais, chamado de programação funcional. A programação funcional enfatiza a avaliação de expressões, ao invés da execução de comandos. As expressões nessas linguagens são formadas utilizando-se funções para combinar valores básicos.

As linguagens funcionais mais conhecidas são o LISP e o Prolog. A linguagem Scheme também é frequentemente citada, por ser uma variante simplificada do LISP. Diversas outras linguagens funcionais são encontradas na literatura, por exemplo, ASpecT, Caml, Clean, Erlang, FP, Gofer, Haskell, Hope, Hugs, Id, IFP, J, Miranda, ML, NESL, OPAL,e Sisal.

5.1 Prolog

O Prolog é uma linguagem de programação prática e eficiente, introduzida em 1973 por Alain Colmerauer e seus associados na Universidade de Marseille, com o propósito inicial de traduzir linguagens naturais. Em 1977, David Warren da Universidade de Edimburgo, implementou uma eficiente versão do Prolog, batizada de Prolog-10. A partir daí, tornou-se uma escolha natural para a resolução de problemas que envolvem a representação simbólica de objetos e relações entre objetos. O fundamento básico por trás do Prolog é a noção de programação em lógica, onde o processo de computação pode ser visto como uma sequência lógica de inferências. Esta noção desenvolve-se então de modo a resultar em um mecanismo automático de prova de teoremas. Atualmente, o Prolog é utilizado em diversas aplicações na área de computação simbólica, incluindo-se aí: bases de dados relacionais, sistemas especialistas, lógica matemática, prova automática de teoremas, resolução de problemas abstratos e geração de planos, processamento de linguagem natural, projeto de arquiteturas, logística, resolução de equações simbólicas, construção de compiladores, análise bioquímica e projeto de fármacos.

O Prolog é uma linguagem diferente, mas de uma simplicidade marcante. Essa diferença decorre do fato de o Prolog ser uma linguagem funcional, e não uma linguagem procedural. Sua estrutura conceitual está intimamente ligada com a lógica matemática, o que a torna uma ferramenta indispensável para o estudo de lógica. Versões diferentes do Prolog podem ser encontradas nas mais diversas plataformas, tanto na forma de compiladores como de interpretadores.

5.2 LISP

O LISP (LISt Processor) é uma linguagem de programação que foi desenvolvida utilizando-se idéias oriundas do estudo da inteligência artificial. Sua constituição é tal que programas escritos em LISP tendem a ser um modelo que emula as habilidades cognitivas do ser humano. O elemento básico utilizado pelo LISP são os símbolos, eminentemente símbolos alfanuméricos.

A partir de 1984, o LISP começou a desenvolver-se em diversos dialetos, sem que um deles dominasse marcadamente os outros. Em um esforço conjunto de

um seleto grupo de pesquisadores em linguagens de programação, representando diversas instituições, desenvolveu-se o Common Lisp, integrando com sucesso diversas características positivas das diferentes versões do Lisp que circulavam na época.

O Common Lisp é um dialeto do LISP, sucessor do MacLisp e influenciado marcadamente pelo Zetalisp e de certa forma também pelo Scheme e pelo Interlisp. O Common Lisp é hoje utilizado como um padrão comercial para a linguagem Lisp, sendo suportado por um grande número de companhias. Posteriormente, o Common Lisp foi enriquecido pela adição de funções genéricas, dando origem a uma variante sua com capacidade de programação orientada a objetos chamada CLOS (Common Lisp Object System).

5.3 Scheme

A linguagem Scheme é um dialeto do LISP que enfatiza a elegância conceitual e a simplicidade. É definida pelo padrão IEEE P1178, sendo substancialmente menor que o Common Lisp. Sua especificação tem aproximadamente 50 páginas, quando comparadas às 1300 páginas do Common Lisp. O Scheme é frequentemente utilizado no estudo de linguagens de programação, devido à sua habilidade em representar várias abstrações de programação com um conjunto bem limitado de primitivas.

6. Linguagens Orientadas a Objetos

As linguagens orientadas a objeto foram originadas a partir da necessidade de se organizar o processo de programação em uma linguagem. A programação orientada a objetos é por si, só uma técnica. Uma linguagem é dita uma linguagem orientada a objetos, se ela suporta o estilo de programação orientada a objetos.

Um dos objetivos da engenharia de software é particionar o projeto de um programa em diversos módulos, de tal forma que o desenvolvimento do software e sua manutenção possam ser implementados com baixo custo. No passado, os paradigmas da engenharia de software derivavam os módulos baseados na funcionalidade de um sistema. Esses módulos correspondiam basicamente a módulos procedimentais, que eram alimentados por dados, gerando novos dados. O paradigma de orientação a objeto mudou essa concepção, idealizando a idéia de objetos, como módulos que se comunicam por meio de mensagens, encapsulando ao mesmo tempo dados e funções, por meio de um mecanismo conhecido como tipo de dados abstratos.

A primeira linguagem orientada a objetos que se tem notícia foi o Simula, desenvolvido em 1967. Posteriormente veio o Smalltalk em 1970. Atualmente, existe uma enorme diversidade de linguagens orientadas a objeto, abrangendo desde linguagens de propósito geral, até linguagens para multimídia e programação em lógica.

6.1 Simula

A linguagem Simula foi desenvolvida por O-J Dahl, B. Myhrhaug e K. Nygaard, no Centro Norueguês de Computação, em Oslo, em 1967, como uma extensão do ALGOL 60. A linguagem Simula mantém algumas propriedades do ALGOL 60, além de extensões tais como os conceitos de classe e co-rotina, bem

como os conceitos associados de variável de referência, manipulação de textos e facilidades de entrada/saída.

Um dos problemas com o Simula é que ela não provê proteção para uma classe, o que foi resolvido posteriormente no Smalltalk.

6.2 Smalltalk

A linguagem Smalltalk foi desenvolvida em 1970. No Smalltalk, não se chama uma função, mas manda-se uma mensagem para um objeto. Dessa maneira, nada do mundo externo pode enxergar a definição de uma classe, o que protege essa classe de qualquer interferência.

Um fator característico do Smalltalk é que qualquer dado é visto como um objeto, mesmo um número. Assim, até mesmo cálculos aritméticos simples beneficiam-se da técnica de orientação a objeto.

Entretanto, mandar mensagens para objetos pode significar um aumento de custo computacional que as vezes se torna um desperdício. Algumas otimizações foram efetuadas, portanto, para evitar a passagem de mensagens em funções simples, tais como operações aritméticas.

6.3 C++

O C++ é uma linguagem de programação de propósito geral, desenvolvida por Bjarne Stroustrup, nos laboratórios da AT&T Bell, no início dos anos 80, como uma evolução do C, incorporando, dentre outras, as seguintes extensões: suporte para a criação e uso de tipos de dados abstratos, suporte ao paradigma de programação orientada a objeto, além de diversos outras pequenas melhorias nas construções existentes no C. Algumas de suas características são o uso de tipos estáticos, a definição de classes, funções virtuais e overload de operadores para o suporte à programação orientada a objeto, o uso de templates para programação genérica, além de prover facilidades de programação de baixo nível (a exemplo do C).

Depois de seu “release” inicial pela AT&T, em 1985, diversas implementações do C++ tornaram-se disponíveis, em mais de 24 tipos de sistemas, desde PCs até mainframes. Em 1989, o número de usuários e diferentes implementações do C++ demandaram a geração de uma norma, sem a qual seria inevitável o surgimento de diversos dialetos. Em 1995 os comitês de normalização da ANSI e ISO C++ chegaram a um nível de estabilidade na linguagem. Basicamente, o C++ corresponde à linguagem descrita em “The C++ Programming Language (2nd edition)” , de Bjarne Stroustrup, ed. Addison Wesley, 1991. A norma que regula o C++ está atualmente em fase de “draft”, sendo catalogada como X3J16/95-0087 pela ANSI e WG21/NO687 pela ISO.

6.4 Objective-C

Objective-C é uma linguagem de programação orientada a objetos, que corresponde a um superconjunto do ANSI-C, provendo classes e passagem de mensagens de um modo similar ao Smalltalk. A linguagem inclui, quando comparada ao C, alguns termos e construções adicionais. Por ser também uma linguagem derivada do C, assim como o C++, ela é frequentemente comparada a esta. O desenvolvimento do C++ se baseia no estilo de programação orientada a objetos iniciado pela linguagem Simula 67. O Objective-C, por seu lado, é originário do estilo de programação encontrado no Smalltalk. No C++, o tipo

estático de um objeto determina quando ele pode receber uma mensagem, ao passo que no Objective-C, seu tipo dinâmico é utilizado. O estilo de programação oriundo do Simula 67 é mais seguro, uma vez que mais erros podem ser detectados ainda em tempo de compilação. O estilo desenvolvido pelo Smalltalk é mais flexível.

Algumas das vantagens do Objective-C incluem a possibilidade de se carregar as definições das classes e métodos em tempo de execução, o fato de os objetos serem tipados dinamicamente, a possibilidade de se utilizar objetos remotos, a persistência e a existência de protocolos de delegação e meta-ação. Alguns de seus principais problemas são a inexistência de herança múltipla e a inexistência de variáveis de classe.

6.5 Java

A linguagem Java foi desenvolvida a partir de 1990 pela Sun Microsystems, como uma linguagem que pudesse executar o mesmo programa em múltiplas plataformas de hardware e software. Seu uso imediato seria a execução de programas pela Internet. Para tanto, não poderia haver nenhum vínculo da linguagem com o hardware e/ou o sistema operacional utilizado, de modo que em princípio qualquer computador conectado à Internet e capaz de entender a linguagem, fosse capaz de executar os programas. Outra especificação básica da linguagem seria a segurança, ou seja, como as máquinas executando os programas em Java estariam em princípio executando programas sem garantias de confiabilidade e procedência, um programa em Java não poderia de modo algum influir na execução de outros programas e o próprio sistema operacional. Com isso, alguns recursos como alocação dinâmica de memória e acesso a arquivos foram sistematicamente eliminados. A linguagem Java foi desenvolvida a partir de um subconjunto do C++, gerando uma linguagem que originalmente foi chamada de Oak. Em 1995, a linguagem foi re-batizada como Java e introduzida na comunidade Internet.

O Java é uma linguagem parcialmente compilada e parcialmente interpretada. Um compilador Java transforma o programa fonte, escrito em Java, em arquivos-objeto chamados *bytecodes.* Esses *bytecodes* precisam ser executados então por interpretadores Java que são desenvolvidos para cada plataforma de hardware/software. Os *bytecodes* podem ser basicamente de dois tipos. O primeiro tipo tem acesso completo à máquina, ou seja, é capaz de manipular a memória, o console e o sistema de arquivos. Programas desse tipo são chamadas de **aplicações** Java. O segundo tipo de *bytecode* sofre uma série de restrições quanto ao acesso de memória, console e sistema de arquivos. Essas restrições são colocadas em nome da segurança, visto que seu destino é a elaboração de programas que serão distribuídos pela Internet, e por isso não provém de fonte conhecida ou confiável. Esses *bytecodes* são chamados de Java **applets**. Os applets têm uma capacidade de atuação bem restrita, de modo que não possam causar danos ao sistema durante sua execução. Normalmente sua atuação se restringe a animações e interações mínimas com o usuário, sendo que seu uso é marcadamente em *browsers* do WWW (World Wide Web). Para executar applets, os browsers necessitam interpretar os bytecodes Java. Browsers com essa capacidade são chamados de *Java aware*. Uma outra variante da linguagem Java é o JavaScript, que são programas colocados em forma de código fonte, incluídos nos textos das páginas HTML. Programas escritos com o

JavaScript não precisam ser compilados, nem geram bytecodes, sendo interpretados diretamente pelos browsers quando a página HTML é interpretada.

6.6 CLOS

CLOS (Common Lisp Object System) é um padrão de programação orientada a objetos desenvolvido a partir do Common Lisp. A maioria das implementações do Common Lisp incluem atualmente uma implementação também do CLOS, por exemplo: MCL, {A}KCL, Allegro CL (incluíndo-se Allegro CL\PC), Ibuki, Lucid, Medley, Symbolics Genera, CLOE, e Harlequin LispWorks.

6.7 Eiffel

A linguagem Eiffel é uma avançada linguagem de programação orientada a objetos, que enfatiza o projeto e a construção de "software" reutilizável de alta qualidade. A linguagem Eiffel não corresponde a um sub-conjunto ou extensão de nenhuma outra linguagem. É uma linguagem que enfatiza o uso de técnicas de programação orientada a objeto, impedindo algumas práticas que podem ser potencialmente perigosas, e que são encontradas em outras linguagens. Entretanto, ela permite um interfaceamento com outras linguagens, tais como o C ou o C++. A linguagem Eiffel suporta o conceito de "design by contract", para garantir a consistência do software gerado.

A linguagem Eiffel foi criada por Bertrand Meyer, e desenvolvida por sua companhia, a "Interactive Software Engineering (ISE)". Seu design é inspirado por algumas preocupações levantadas por engenheiros de software, diante da criação de sistemas mais complexos. Como linguagem, vem evoluindo continuamente, desde sua concepção, em 1986.

Compiladores para a linguagem Eiffel podem ser encontrados para diversos sistemas operacionais e plataformas de hardware, destacando-se as seguintes:

PC: DOS, OS/2, Windows 3.1, Windows 95, Windows NT, PC Unix (Interactive, SCO, and ESIX), Nextstep, Linux

Outras plataformas de Hardware: Sparc (SunOS & Solaris), NeXTStep, HP9000, DEC 5xxx, Sony News, DG Aviion, DEC Alpha OSF-1, DEC OpenVMS, RS6000, Pyramid, QNX, Silicon Graphics, Macintosh (Motorola & PowerPC)

7. Linguagens para Aplicações Específicas

7.1 Linguagens para Bancos de Dados

7.1.1 Clipper

O CA-Clipper é uma linguagem de programação para aplicações em bancos de dados, desenvolvida e comercializada pela Computer Associates, que corresponde a um superconjunto do dBASE III+, incorporando também algumas características do C/C++ e Smalltalk.

7.1.2 SQL

SQL é um acrônimo para “Structured Query Language”. O SQL é uma linguagem de acesso a banco de dados originária de plataformas de grande porte (mainframes e mini-computadores), que entretanto vem se tornando popular em PC´s, principalmente em ambientes do tipo cliente-servidor.

O SQL possui um núcleo comum, que geralmente se encontra disponível em todos os produtos que implementam a linguagem. Esse núcleo provê os fundamentos básicos para a criação e o uso de bases de dados. Entretanto, diferentes implementações incorporam extensões ao SQL, colocadas com o intuito de diferenciar essas implementações para certos tipos de aplicações.

O SQL é baseado no modelo de base de dados relacional, desenvolvido por E.F. Codd em 1970, nos laboratórios da IBM. Nos anos subsequentes esse modelo foi refinado por C.J. Date, também da IBM.

Em 1992, colocaram-se duas versões para uma norma definindo o SQL, a ISO/IEC 9075:1992, “Information Technology - Database Languages - SQL” e a ANSI X3.135-1992, “Database Language SQL” . Essas normas representam o chamado SQL´92.

Em 1993, tanto a ANSI como o ISO decidiram dividir o desenvolvimento do SQL em uma norma multi-partes: Framework, Foundation, SQL/CLI, SQL/PSM, SQL/Bindings, SQL/XA e SQL/Temporal. Essa norma multi-partes foi então designada SQL3, estando, entretanto, ainda incompleta.

Os esforços no desenvolvimento da linguagem concentram-se no momento em definir uma linguagem computacional completa para a definição e gerenciamento de objetos complexos e persistentes. Isso inclui: hierarquias de generalização e especialização, herança múltipla, tipos definidos pelo usuário, triggers e asserções, suporte a sistemas baseados em conhecimento, expressões recursivas de busca, além de ferramentas adicionais de administração de dados. Inclui ainda a especificação de tipos abstratos de dados, identificadores de objetos, métodos, herança, polimorfismo, encapsulamento e todas as outras facilidades normalmente associadas com o gerenciamento de objetos. A expectativa para o término do SQL3 é no momento para 1999.

7.2 Linguagens para Simulação

7.2.1 MATLAB

O MATLAB (MATrix LABoratory) é uma linguagem desenvolvida pela MathWorks, voltada para aplicações científicas, eminentemente para simulações de sistemas. É uma linguagem estruturada bem versátil, de sintaxe simples, mas com todas as estruturas de controle usuais encontradas em uma linguagem procedural. Sua maior característica é a implementação de uma sofisticada e eficiente biblioteca de funções, envolvendo matrizes e funções de “plotting”, que fazem com que se torne muito apropriada em cálculos científicos que exigem uma visualização de resultados.

7.3 Linguagens de Scripts

7.3.1 Tcl/Tk

O TCL (Tool Command Language) é uma linguagem desenvolvida por John Ousterhout, em 1990, na Universidade da Califórnia, Berkeley. O TK é um toolkit para o desenvolvimento de GUI´s, baseado no X-Windows, que fornece widgets com visual semelhante ao Motif. Juntos, eles são conhecidos como TCL/TK, e correspondem a uma linguagem de script para o desenvolvimento de interfaces gráficas em aplicações. Entretanto, o TCL suporta muitas das características de linguagens procedurais convencionais, incluindo-se o uso de variáveis, chamadas de procedimentos e estruturas de controle. O TCL/TK é muito utilizado, devido a ser uma ferramenta amigável para a geração de interfaces gráficas. Uma das vantagens do TCL/TK é que ele pode ser utilizado conjuntamente com outras linguagens de programação, por exemplo o C. Com isso, implementa-se a interface gráfica em TCL/TK e partes mais sofisticadas do programa em outras linguagens mais adequadas.

7.3.2 Perl

A linguagem PERL (Practical Extraction and Report Language) é uma linguagem interpretada, desenvolvida por Larry Wall, otimizada para a varredura de arquivos de texto, extraindo informações destes, e imprimindo "reports" baseados nessas informações. É tambem uma boa linguagem para diversas tarefas de administração de sistemas. A linguagem foi criada com a intenção de ser mais prática (fácil de usar, eficiente e completa) do que bonita (pequena, elegante ou mínima). Ela combina diversas características do C, sed, awk e sh, de modo que programadores com familiaridades nessas linguagens devem encontrar pouca dificuldade. A sintaxe das expressões é bem próxima do C. O PERL, ao contrário de outras linguagens de script, não limita o uso da memória (está limitado à memória física), permite recursão ilimitada e suas tabelas hash, usadas por arrays associados, pode crescer tanto quanto o necessário, de modo a evitar a degradação do desempenho.

O PERL usa uma técnica sofisticada de "pattern matching", de modo que pode varrer rapidamente pacotes grandes de dados. Apesar de ser otimizada para a varredura de texto, o PERL pode lidar também com dados binários.

7.4 Linguagens de Formatação de Textos

7.4.1 TeX/LaTeX

O TeX é uma linguagem para editoração eletrônica criada por Donald E. Knuth, que é baseada na definição de macros. Um conjunto mínimo de macros é conhecido como "plain TeX", sendo utilizado como base para a construção de "packages" mais sofisticados. O LaTeX é um "package" de macros em TeX, originalmente desenvolvido por Leslie Lamport em 1985, que estrutura um documento por meio de parágrafos lógicos, permitindo ao autor se concentrar mais no texto do que na forma de sua apresentação. Uma das características que tornou o LaTeX muito popular é a sua flexibilidade para a descrição de fórmulas matemáticas.

7.4.2 HTML

O HTML (Hyper Text Markup Language) é uma linguagem de "markups", que permite a formatação de texto e imagens, utilizada para a construção de páginas hiper-mídia na Internet. Junto com o protocolo de rede http, foi um dos responsáveis pela explosão do WWW (World Wide Web), que popularizou a Internet mundo afora. O HTML é uma aplicação do SGML (Standard Generalized Markup Language), e devido à sua popularidade, diversos conversores "de" e "para" HTML foram desenvolvidos para os mais diversos aplicativos envolvendo textos, gráficos e imagens.

O HTML encontra-se já na sua versão 3.2. A versão 3.0 (proposta) foi abandonada por ser complexa demais para padronização imediata. A versão atual é portanto, neste sentido, mais "modesta" que a 3.0 (por exemplo, deixando de incluir suporte a equações matemáticas).

7.4.3 Postscript

Postscript é uma linguagem de programação otimizada para a impressão de gráficos e textos. Dentro do jargão utilizado, trata-se de uma "linguagem de descrição de página". O Postscript foi introduzido pela Adobe em 1985, aparecendo inicialmente na impressora Apple LaserWriter. O propósito básico do Postscript foi a criação de uma linguagem conveniente para a descrição de imagens em um modo independente de dispositivo. Essa independência diz respeito ao fato da imagem ser descrita sem nenhuma referência a características específicas de impressoras ou outros dispositivos de visualização. Desse modo, programas de editoração de texto podem gerar arquivos Postscript, que nada mais são do que um arquivo de comandos na linguagem Postscript, que podem ser enviados diretamente para impressoras capazes de interpretar o Postscript.

A linguagem Postscript inclui ainda um conjunto padrão de fontes, chamadas de fontes Postscript, que são eventualmente utilizadas por outras linguagens de programação para exibição de textos em consoles gráficos.

Devido à necessidade de inserir imagens codificadas via Postscript dentro de páginas Postscript, surgiu o Postscript encapsulado (também conhecido como EPS), que permite essa inserção.

7.4.4 PDF

O PDF (Portable Data Format) é um aperfeiçoamento da idéia inicial do Postscript, corrigindo alguns de seus defeitos, e enfatizando algumas características cujo crescimento da Internet e do WWW demandou. O PDF, assim como o Postscript foi desenvolvido pela Adobe, e sua maior vantagem em relação ao Postscript é a facilidade de manipulação e busca em textos previamente formatados. O PDF vem a ser uma tentativa de superar os problemas intrínsecos do Postscript e do HTML, no contexto de geração de documentos hipermídia na Internet. Ele mantém o mesmo poder descritivo do Postscript, e a mesma agilidade e poder de manipulação do HTML.

Paralelo a algumas iniciativas como o HTML versão 3, que busca aprimorar a linguagem padrão de hipermídia, a Adobe tenta vender o PDF como uma alternativa ao HTML versão 2, e luta por tentar colocá-lo como um padrão de desenvolvimento de hipermídia para o WWW.

8. Linguagens de Programação Visual

As linguagens de programação visual partem do princípio de que gráficos são mais fáceis de serem entendidos do que textos. Sendo assim, a especificação de um programa por meio de diagramas e outros recursos gráficos tende a tornar a própria programação mais fácil, permitindo mesmo que usuários sem muitas habilidades em programação gerem programas.

Uma representação gráfica de um programa tende a ser uma descrição de alto nível para o funcionamento do programa. Este tipo de representação normalmente limita bastante a flexibilidade dos programas que podem ser desenvolvidos, mas permite que rápida e facilmente se elaborem programas, dentro de um escopo limitado de opções. Com isso, as aplicações alvo para a programação visual se restringem a aplicações bem específicas, que podem se beneficiar deste tipo de paradigma, não sendo, tais linguagens, linguagens de programação para propósitos gerais.

8.1 Ferramentas de Programação Visual

Nem todo uso de gráficos e diagramas visuais em programação podem ser chamados de linguagens visuais. Em alguns casos, o mais adequado é chamá-los de ferramentas de programação visual. Nesse caso, o uso de gráficos e diagramas não especifica diretamente um programa a ser executado, mas é uma ferramenta de auxílio na elaboração de tais programas. Por exemplo, os toolkits de elaboração de interface, que permitem ao usuário especificar diferentes controles em janelas de interface do programa, de forma visual, encontram-se nesta categoria.

8.2 Linguagens de Programação Visual Híbridas

Nessa categoria, incluem-se as linguagens de programação que não são somente visuais, mas têm parte de sua especificação determinada de forma visual. Normalmente este tipo de linguagem de programação visual não é simplesmente uma “linguagem”, mas um “produto”, que é comercializado por uma empresa de software. Em alguns casos esse “produto” está mais para uma linguagem de programação convencional (por textos), dotada de diversas ferramentas de programação visual, do que realmente linguagens de programação visual. Em outros casos, as ferramentas de programação visual efetivamente participam da elaboração do programa, como por exemplo na determinação de hierarquia de classes e métodos, em linguagens orientadas a objeto. As linguagens que incluem-se nesta categoria são normalmente desenvolvimentos de linguagens de programação convencional, acrescidas das ferramentas visuais que lhes dão o “status” de visuais. Exemplos incluem o Delphi, o Visual Basic, o Visual C++.

8.3 Linguagens de Programação Visual Puras

As linguagens de programação visual puras são aquelas em que o programa é determinado exclusivamente por meio dos gráficos e diagramas que o especificam. O paradigma mais eficiente nesse tipo de programação é a elaboração de diagramas com nós e arcos, sendo que os nós representam módulos de software, a partir de uma biblioteca prévia de módulos, e os arcos determinam a conexão entre os diferentes módulos. Como esses programas dependem fortemente dos módulos de software que existem previamente no sistema, seu uso é limitado a

aplicações bem definidas. Alguns dos mais bem sucedidos exemplos de linguagens de programação visual puras são dados a seguir.

8.3.1 Khoros

O Khoros é uma linguagem de programação visual utilizada principalmente em computação gráfica. Na verdade o Khoros é mais do que simplesmente uma linguagem, mas todo um "ambiente" de programação gráfica, que permite ao programador efetuar o processamento de imagens e análises sofisticadas por meio da elaboração de diagramas com nós e arcos. O Khoros vem sendo utilizado com resultados muito positivos tanto para a análise como síntese de imagens gráficas, principalmente nas áreas médicas. Entretanto, por ter uma especificação aberta, pode ser utilizado em princípio em qualquer outro tipo de aplicação.

8.3.2 Simulink

O Simulink é outra linguagem de programação visual, utilizada eminentemente na simulação de sistemas dinâmicos. O Simulink opera em conjunto com o Matlab, de modo a permitir a especificação de diferentes sistemas dinâmicos, determinar as condições e parâmetros da simulação, e efetivamente executá-la. Permite ainda a análise dos resultados da simulação. De modo análogo ao Khoros, o Simulink provê uma biblioteca de módulos básicos, que podem ser utilizados na elaboração de diagramas que especificam o sistema, a simulação e a análise a ser efetuada. Os módulos são conectados por arcos que determinam o relacionamento entre os diferentes módulos. Módulos sofisticados podem ser construídos e transformados também em módulos, podendo então ser referenciados por outros programas. Por se utilizar do Matlab, novos módulos podem ser construídos, a partir do Matlab, ou então em linguagem C.