Uma Proposta de Linguagem de Programação para Robótica

Gustavo Vasconcelos Arnold[1*], **Pedro Rangel Henriques**[1] , **Jaime Cruz Fonseca**[2]

[1]Departamento de Informática – Universidade do Minho
Campus de Gualtar – 4710-057 – Braga – Portugal

{gva,prh}@di.uminho.pt

[2]Departamento de Electrónica Industrial – Universidade do Minho
Campus de Gualtar – 4710-057 – Braga – Portugal

jaime.fonseca@dei.uminho.pt

Abstract. *This paper presents some existing problems in robotics programming languages and a possible solution using a reactive and synchronous language called RS. An implementation will show how the RS language can facilitate the robot´s programming, in this case, industrial robots. However, it should be necessary to create an intermediate module responsible for translating a declarative language to assembly code, in different robots, granting code portability.*

Resumo. *Este artigo destina-se a apresentar alguns problemas existentes nas linguagens de programação de robôs, e uma possível solução dos mesmos através da utilização da linguagem Reativa Síncrona RS. Uma implementação prática será apresentada a fim de mostrar como o uso desta linguagem pode facilitar o desenvolvimento de programas para os robôs, neste caso, industriais. Entretanto, verifica-se a necessidade da existência de um módulo intermediário responsável por traduzir uma linguagem declarativa para código final de diferentes robôs, a fim de garantir a portabilidade de código.*

1. Introdução

O uso de robôs em aplicações industriais tem aumentado nos últimos anos. Os robôs deixaram de ser privilégio de grandes indústrias e têm-se disseminado em diferentes tipos de indústrias e aplicações. Devido a esta grande quantidade de aplicações, robôs de vários tipos, de vários fabricantes, são utilizados atualmente.

No entanto, a programação de robôs é, regra geral, ainda realizada a baixo nível, senão mesmo em código final, em linguagens estruturadas imperativas. Assim sendo, esta tarefa torna-se demorada e dispendiosa, sendo, portanto, pouco lucrativa.

A agravar o estilo de programação usado, registra-se o fato dessa programação ser totalmente dirigida para o equipamento específico, em causa em cada momento. Isto é, mudando-se o modelo, ou a marca, do robô usado é necessário reescrever o programa,

*Docente afastado da Universidade Católica do Salvador - Brasil, temporariamente na Universidade do Minho - Portugal, para obtenção do grau de Doutor; Bolsista FCT referência 074/BI-M/01 (projeto Metamedia)

implicando novo tempo de aprendizagem. Nestas circunstâncias verifica-se vulgarmente que pouco código pode ser reaproveitado, o que agrava ainda mais a já baixa falta de produtividade.

Este artigo apresenta as formas de programação de robôs, juntamente com suas características e problemas existentes, no que diz respeito a engenharia de software, e propõe uma solução para estes problemas, através de uma linguagem declarativa. Para validar esta proposta, será apresentado um compilador, que resolve quase todos os problemas. O único problema ainda não resolvido diz respeito à portabilidade de código fonte, problema esse que dará origem à proposta de trabalho futuro apresentada neste artigo.

Um passo já dado — concretamente, na programação do robô NACHI SC15F [Piola, 1998] — é a utilização de uma linguagem de mais alto nível, a linguagem Reativa Síncrona RS [Toscani, 1993], com um caráter declarativo, baseada em um sistema de regras reativas (condição => ação). Com esta solução, melhora-se o tempo/esforço de programação, criando um mais alto nível de abstração no desenvolvimento de programas.

Para usar a linguagem RS no controle deste robô foi necessário implementar um compilador capaz de traduzir o autômato gerado pelo processador da linguagem RS (autômato simples e eficiente) em código para o robô em questão.

Para melhor apresentar este trabalho, o artigo foi estruturado da seguinte forma:

- Na seção 2 serão apresentados o conceito de robô, bem como as formas, vantagens/dificuldades de sua programação e um exemplo de linguagem de programação de um robô;
- Na seção 3 serão apresentadas as principais características da linguagem RS, que a tornam uma linguagem adequada para o desenvolvimento de sistemas robóticos;
- Na seção 4 será apresentado um compilador que torna possível a programação de um robô através da linguagem RS, juntamente com um exemplo de programa;
- Na seção 5 serão apresentadas as conclusões, bem como uma proposta a ser desenvolvida no futuro.

2. Programação de Robôs Industriais

Antes de verificar as características, vantagens e dificuldades da programação de robôs, é importante apresentar o conceito, isto é, explicar o que vem a ser um robô.

2.1. O que é um Robô?

Um robô é um dispositivo eletro-mecânico, com atuador mecânico controlado por computador, o qual pode ser programado para realizar automaticamente uma grande variedade de tarefas. A Robotic Industries Association [Rehg, 1997] define um robô como:

"Um sistema de robô industrial inclui o robô (hardware e software) consistindo do: manipulador, fonte de energia e controlador; órgãos terminais; quaisquer equipamentos, dispositivos e sensores aos quais o robô está ligado para realizar suas tarefas; e qualquer interface de comunicação entre o robô e o sistema que o está operando e monitorando."

2.2. Formas de Programar um Robô

Um robô normalmente é programado através da inclusão de comandos na memória de seu controlador. Existem quatro métodos de programação de robôs [Groover, 1987]:

- *Setup* manual - nesta forma, a ”programação”é feita através da colocação de obstáculos físicos que são utilizados como barreiras para controlar os pontos finais de cada movimento. No entanto, a simples colocação de obstáculos não corresponde precisamente a um método de programação, e sim a um controle mecânico;
- Programação *leadthrough* - o programador precisa movimentar o robô de acordo com o movimento desejado durante um processo de aprendizado, incluindo, desta forma, o programa na memória do controlador. Apesar de ser um método simples de programar, possui algumas desvantagens como: a necessidade de interromper o processo de produção e esperar a programação; ser limitado em termos de incorporar mecanismos de tomada de decisão no programa; e não ser compatível com tecnologias de computação existentes (CAD/CAM, comunicação, banco de dados);
- Programação direta em Linguagens de programação próprias - semelhante a programação de um computador, respeitando-se as características do robô. Tem várias vantagens como: um melhor tratamento dos sensores; capacidade de controlar dispositivos externos; aumento da capacidade de expressar a lógica do programa; e possibilidade de comunicar com sistemas computacionais externos, acedendo a vários recursos como bases de dados ou bases de conhecimento;
- Programação *off-line* - semelhante à anterior, utiliza-se uma linguagem de programação própria (com as vantagens já indicadas), mas o desenvolvimento não é feito diretamente sobre o robô; em vez disso é feito num sistema de computação normal, permitindo simular o programa, e somente quando pronto e correto, o programa será enviado ao robô para execução. A grande vantagem deste método é o fato de não ser necessário interromper o processo de produção para construir e testar o programa.

Como pode ser visto, o uso de uma linguagem de programação, off-line ou não, traz muitas vantagens para a área de robótica. No entanto, estas linguagens de programação apresentam algumas características que prejudicam o desenvolvimento de sistemas robóticos, no que diz respeito a engenharia de software:

- As linguagens utilizadas são do paradigma imperativo, e algumas delas são do tipo assembly, ou seja, as linguagens utilizadas pertencem a um nível próximo da máquina, o que faz com que o programador desvie sua atenção da lógica do programa para aspectos do hardware;
- O controle do fluxo de execução é realizado através de desvios incondicionais (*goto*). Mesmo nos desvios condicionais a execução é desviada para um ponto qualquer do programa definido por um rótulo. Isto prejudica os critérios da programação estruturada;
- Não existe portabilidade de código fonte, pois cada linguagem de programação é específica para um robô;
- Como consequêcia do item anterior, a reutilização só é possível entre programas feitos para as mesmas máquinas;

Figura 1: Robô NACHI SC15F

- Devido a não portabilidade de código fonte, os programadores precisam ser treinados sempre que houver alguma troca de equipamento;

Existem algumas linguagens de programação de robôs de alto nível, fáceis de usar, visuais. Mesmo assim, estas linguagens continuam sendo específicas de um único robô, o que traz a série de consequências que já foram vistas acima.

2.3. Marcação de Pontos

A principal característica que diferencia as linguagens de programação de robôs das linguagens de programação convencionais é a necessidade de marcar pontos.

Isto é preciso porque os robôs precisam se movimentar de lugar, e em cada lugar eles deverão fazer alguma tarefa (soldar, pintar, pegar, largar, desviar, etc). Cada um destes lugares possui um ponto associado a ele.

Neste projeto, foi utilizado, como robô, um braço mecânico, como pode ser visto na figura 1. Para posicionar este braço em um ponto é preciso utilizar seis coordenadas: as três primeiras, (X,Y,Z) são responsáveis por posicionar o robô espacialmente em algum lugar em três dimensões; enquanto que as outras três coordenadas (R,P,Ya) são responsáveis por posicionar o punho do robô.

Os movimentos possíveis para o punho são: *Roll* (R), movimento semelhante ao usado para dizer ”mais ou menos”; *Pitch* (P) movimento semelhante ao usado para dizer ”até logo”; e *Yaw* (Ya) movimento semelhante ao usado para dizer ”não”.

Para realizar a marcação de pontos, normalmente se utiliza um dispositivo chamado *teach box*, dispositivo parecido com um joystick, responsável por controlar os movimentos do robô. Desta forma, através do *teach box* se movimenta o robô, e quando o mesmo estiver localizado no lugar desejado, grava-se este ponto.

Esta marcação de pontos pode ser feita antes ou depois do desenvolvimento do programa.

2.4. Linguagem de Programação do Robô NACHI SC15F

O robô utilizado neste projeto foi o robô NACHI SC15F, um robô para propósitos gerais, fabricado pela companhia japonesa Nachi-Fujikoshi, e que pode ser visto na figura 1.

A linguagem de programação do robô NACHI SC15F é semelhante a linguagem BASIC. Para desenvolver programas através desta linguagem é necessário gravar um arquivo com os pontos onde o robô deverá passar durante sua movimentação. Esta gravação é feita através do *teach box* do robô, da forma como foi dito na seção anterior.

As principais características desta linguagem são [Corporation, 1994]:

- As linhas do programa precisam ser numeradas, para indicar a ordem de execução do programa. Esta numeração varia de 1 à 9999, onde cada linha corresponde a um único comando;
- As constantes númericas aceitas são: inteiro, real, hexadecimal, binário, e uma constante que expressa um ângulo em graus;
- As constantes ponto indicam uma localização para o robô. O formato é (X, Y, Z, R, P, Ya), onde X, Y e Z correspondem as coordenadas da base do robô, e R, P e Ya correspondem as coordenadas do punho do robô, sendo R o ângulo de *Roll*, P o ângulo de *Pitch*, e Ya o ângulo de *Yaw*;
- As variáveis possuem nomes pré-definidos, sendo os nomes dependentes dos tipos das variáveis. As variáveis do tipo inteiro possuem o formato Vn% ou V%[n], onde n pode variar de 1 até 200 (Ex: V1% = 10, V%[2] = 20). As variáveis do tipo real possuem o formato Vn! ou V![n] onde n pode variar de 1 até 100 (Ex: V1! = 10,5, V![2] = 20 * V1!). As variáveis do tipo caracter e string possuem o formato Vn$ ou V$[n], onde n pode variar de 1 até 20 (Ex: V1$ = ”ABC”, V$[2] = ”Z”+ V1$). As variáveis do tipo ponto possuem o formato Pn ou P[n], onde n pode variar de 1 até 999 (Ex: P1 = (100, 0, 100, 0, 0, 90));
- É necessário o uso de rótulos para o controle de fluxo. Um rótulo é definido iniciando-se o nome com um asterisco (*) seguido de um caracter. Os comandos *GOTO* e *IF* necessitam de rótulos para indicar o ponto do programa (comando) onde a execução deve prosseguir;
- O comando *STOP* interrompe a execução do programa, enquanto que o comando *END* termina a execução do mesmo.

3. Linguagem RS

A linguagem RS [Toscani, 1993] tem características que facilitam bastante o desenvolvimento e manutenção de programas para robôs, como se vai mostrar mais à frente. Dentre estas características temos:

- É orientada para a especificação e implementação de sistemas reativos, que são dirigidos por estímulos provenientes de um ambiente externo. Ou seja, sistemas que dependem de estímulos externos para desencadear as ações associadas àquele estímulo;
- É uma linguagem simples, com comandos (regras) do tipo *condição* => *ação* que facilitam a programação e permitem uma maior concentração, por parte do programador, na lógica do sistema que está sendo construído. Esta estrutura de comandos permite aliar a simplicidade ao poder de programação, pois expressa corretamente o raciocínio dos ambientes reativos;
- Os programas podem ser estruturados segundo um conjunto de módulos; e os módulos podem ser estruturados segundo um conjunto de caixas de regras;

- É uma linguagem paralela e distribuída, pois seus módulos podem ser executados em processadores ou máquinas diferentes;
- É uma linguagem de tempo real, pois a mesma adota a hipótese do sincronismo, na qual suas reações são executadas em tempo zero, tornando os sinais de saída síncronos com os sinais de entrada;
- Possui mecanismos para o tratamento de situações de exceção;
- É uma linguagem declarativa, de alto nível.

Dentre os problemas existentes com as linguagens de programação para robôs, o único que a linguagem RS ainda nao trata é o problema da portabilidade, visto que existe apenas um compilador específico implementado para usar esta linguagem em um robô NACHI SC15F. No entanto, esta implementação, da qual será visto um exemplo na seção seguinte, serviu para provar, na prática, que a linguagem RS possui características que melhorariam muito a programação de robôs, definindo então a proposta deste artigo, que pretende criar um módulo capaz de gerar código final para programação de diferentes robôs industriais, a partir do mesmo programa fonte.

Um outro exemplo, apenas simulado, de utilização da linguagem RS no controle de uma célula de manufatura pode ser visto em [Arnold, 2000].

4. Exemplo de Implementação

Tanto neste exemplo como no caso geral, para desenvolver um programa RS para controlar um robô, é preciso:

- identificar quais as entradas digitais existentes no robô que serão necessárias para realizar determinada tarefa e definí-las como sinais de entrada do programa RS;
- especificar um sinal de entrada especial para dar início a execução do autômato, e consequentemente do programa a ser executado pelo robô. Este sinal ocorre no estado pronto do robô, que representa o estado em que o mesmo se encontra ligado e com o programa a ser executado carregado em sua memória;
- identificar que funções deverão ser executadas pelo robô para realizar determinada tarefa corretamente e associá-las a sinais de saída, que serão enviados para o ambiente externo através do comando *emit*. Os sinais de saída poderão conter parâmetros que sejam necessários para executar determinada função.

É necessário especificar pontos por onde o robô deverá passar, no entanto o programador RS não precisa se preocupar com os mesmos, apenas assume que eles já existem, preocupando-se apenas com a lógica do sistema. Os pontos podem ser gravados antes ou depois de feito um programa RS, utilizando-se, por exemplo, uma *teach box*, como foi explicado na seção 2.3.

4.1. Exemplo de programa RS para controlar o robô industrial NACHI SC15F

Um exemplo de programa escrito em RS e utilizado, na prática, em um robô Nachi SC15F pode ser visto na figura (2) [Piola, 1998].

Neste exemplo, o programa apenas controla o robô de forma que o mesmo execute dez ciclos passando por três pontos, `p1, p2` e `p3`. Inicialmente, o robô pode se

```
 rs_prog nr_0001: [input: [ligado, sensor],
 output: [ir(P), fim, ferramenta(C),
          usar(B), deslocar(D,X,Y,Z)],
 module mov:
 [ input: [ligado, sensor],
  output: [ir(P), fim, ferramenta(C),
           usar(B), deslocar(D,X,Y,Z)],
  t_signal: [],
  p_signal: [t1, t2, t3, t4, t5],
  var: [count1],
  initially: [activate(rules), emit(usar(1)),
              emit(ferramenta(0)), count1:=0, up(t1)],
  ligado#[t1] ===> case [count1=10--->[count1:=0,
                         emit(fim),up(t1)],
                     else--->[count1:=count1+1,
                              emit(ir(p1)),up(t2)],
         [t2] ===> [emit(ir(p2)),
                    emit(deslocar(0,0,0,100)), up(t3)],
         [t3] ===> [emit(ir(p3)),
                    emit(deslocar(0,0,0,0)), up(t1)],
  sensor#[t1] ===> [emit(ir(p4)),up(t4)],
         [t4] ===> [emit(ir(p5)),up(t5)],
         [t5] ===> [emit(ir(p6)),up(t1)],
 ]
].
```

Figura 2: Programa RS para controlar o robô.

encontrar em qualquer lugar; geralmente o mesmo se encontrará em uma posição de repouso, afastado do ponto `p1`. A partir desta posição, o robô deverá se mover para o ponto `p1`. Depois, o robô se dirigirá para o ponto `p2`, e em seguida fará um deslocamento em linha reta, definido pelas coordenadas `X, Y, Z`, mantendo a orientação da ferramenta, ou seja, o robô se deslocará até o ponto definido sem alterar a posição final da ferramenta. Após passar por `p2` e enquanto se dirige para `p3` tudo acontece de igual modo, o mesmo sofrerá novamente um deslocamento de forma a retomar as coordenadas `p1` iniciais.

Se, durante a execução deste programa, o sensor `sensor` for acionado, o robô assumirá outro comportamento, durante apenas um ciclo, devendo passar pelos pontos `p4, p5` e `p6`.

Neste programa, a entrada a ser usada corresponde ao sensor `sensor`. O início de atividade será disparado pelo sinal de entrada `ligado`. As funções a serem executadas são:

- `ir(P)`, responsável por fazer o robô se mover para as coordenadas especificadas pelo ponto `P`;
- `fim` responsável por encerrar a execução do programa;
- `ferramenta(C)` responsável pela escolha de uma determinada ferramenta especificada em `C`;
- `usar(B)` responsável por indicar o arquivo de pontos a ser utilizado pelo programa, neste caso o arquivo `B`;
- e `deslocar(D,X, Y, Z)`, responsável por fazer o robô se deslocar até o ponto definido pelas coordenadas `X, Y` e `Z`, mantendo a orientação da ferramenta `D`.

As variáveis utilizadas no programa RS, `t1, t2, t3, t4` e `t5`, não correspondem diretamente aos pontos, mas sim ao seu estado comportamental, o qual representa o ponto onde está e a reação que pode ter. Em termos de controle puro da movimentação do robô, bastaria emitir somente comandos para os respectivos deslocamentos. No entanto, não seria possível testar valores dos sensores; na realidade, é a execução do comando `up` (responsável por ativar os sinais temporários) que força o teste dos sinais, incluindo os sensores.

Quando compilado, este programa RS irá gerar um autômato que modela o comportamento desejado para o robô. Em [Piola, 1998] foi desenvolvido um tradutor capaz de, a partir do autômato gerado pelo compilador de RS, criar o programa a ser executado no robô Nachi SC15F. O programa gerado por este tradutor pode ser visto na figura (3).

Basicamente, o tradutor baseia-se num ciclo padrão que percorre o autômato otimizado gerado, criando rótulos no código para cada estado (correspondente aos sensores utilizados) e substituindo cada comando `emit` pelo código final associado ao sinal de saída deste comando. Para tal, recorre a uma tabela de tradução (previamente construída, a parte) que a cada função (sinal de saída) associa o respectivo código.

Este tradutor tornou possível a utilização da linguagem RS no desenvolvimento de programas para o robô Nachi SC15F, uma vez que sem ele, com o compilador de RS existente, apenas se podia simular o processo (pois só se produzia um autômato que era interpretado pelo simulador). Com este tradutor, passou-se a produzir código final para carregar no robô a controlar, tornando-se o desenvolvimento de programas para robôs

```
10 USE 1
20 TOOL 0
30 V1%=0
40 *S1
41 JMPI 4, I1
50 IF V1%=10 THEN *L2X1 ELSE *L2X2
60 *L2X1
70 V1%=0
80 END
90 GOTO *S1
100 *L2X2
110 V1%=V1%+1
120 MOVE P,P1,T=3
130 MOVE P,P2,T=3
140 SHIFTA 0,0,0,100
150 MOVE P,P3,T=3
160 SHIFTA 0,0,0,0
170 GOTO *S1
180 MOVE P,P4,T=3
190 MOVE P,P5,T=3
200 MOVE P,P6,T=3
210 GOTO *S1
```

Figura 3: Programa gerado para o robô.

mais simples e rápido (abstraindo todos os detalhes físicos específicos), visto que foi possível evitar a programação imperativa e dependente do mesmo.

Um exemplo da não estruturação (que se pretende evitar), que pode existir em programas robóticos, surge no programa da figura (3), na linha 41. Nesta linha, será avaliado se existe algum sinal no sensor `I1`. Se não existir, o programa seguirá o fluxo de execução normal. No entanto, se o sensor contiver algum sinal ativo, o programa sofrerá um desvio de sua execução (`JMPI 4`) para a quarta instrução de movimentação (`MOVE P,P4,T=3`), linha 180, que corresponderá à resposta ao sensor `sensor`, em `t1`. Ou seja, um programa assim codificado torna-se difícil tanto de escrever como de ler (é preciso contar as instruções de movimentação para determinar para aonde será o desvio).

5. Conclusão

Este artigo teve como objetivo apresentar as principais características das linguagens de programação para robôs, bem como as dificuldades existentes, juntamente com as características que tornam a linguagem RS uma linguagem adequada e que facilita este tipo de programação. Para provar esta possibilidade, foi implementado um compilador capaz de transformar o autômato gerado a partir da linguagem RS na linguagem de um robô específico. Esta implementação trouxe algumas conclusões, tanto com relação ao compilador como com relação a trabalhos futuros, que correspondem a proposta apresentada neste artigo.

Uma característica do compilador implementando é a extensibilidade: novos sensores e novas funções podem ser incluídos no sistema, sem precisar alterar o tradutor. Por um lado o programa genérico de tradução trata todos os sensores da mesma maneira, sem necessidade prévia de declaração; quanto a inclusão de novas funções basta implementar cada uma no código do robô e acrescentar na tabela de mapeamento esta nova associação.

Contudo a necessidade de gerar código final específico para cada robô cria a necessidade de modificar todo o tradutor cada vez que se pretende usar equipamento diferente.

Como trabalho futuro, no contexto acabado de descrever, pretende-se pesquisar o tema ”geração de código final para programação de diferentes robôs industriais, a partir do mesmo programa fonte”. Vai-se caracterizar detalhadamente este domínio, o problema da geração de código e sua otimização, de modo a poder-se estudar a viabilidade de um ambiente genérico para resolução sistemática deste tipo de problemas.

Concretamente, é objetivo deste projeto desenvolver um sistema para geração automática dos módulos gerador/otimizador de código que permitirão a construção sistemática de compiladores (usando sempre o mesmo ”front-end”), compiladores esses que traduzam uma linguagem declarativa de programação de robôs para o respectivo código final.

6. Bibliografia

Referências

Arnold, G. V. (2000). Controle de uma célula de manufatura através da linguagem RS estendida. In *Anais do I Congresso de Lógica Aplicada à Tecnologia (LAPTEC'2000)*, pages 145–163.

Corporation, N. F. (1994). *Ar Controller Operating Manual*. Nachi-Fujikoshi Corporation, 1st edition.

Groover, M. P. (1987). *Automation, Production, Systems, and Computer-Integrated Manufacturing*. Prentice Hall, Inc.

Piola, S. J. (1998). *Uso da Linguagem RS no Controle do Robô Nachi SC15F*. Trabalho de Conclusão de Curso de Graduação, Departamento de Informática, UCS, Caxias do Sul, Brasil.

Rehg, J. A. (1997). *Introduction to Robotics in CIM Systems*. Prentice Hall, Inc., 3rd edition.

Toscani, S. S. (1993). *RS: Uma Linguagem para Programação de Núcleos Reactivos*. Tese de doutoramento, Depto de Informática, UNL, Lisboa, Portugal.