International Journal of Soft Computing 1(3): 181-183, 2006


# Mechanisms to Map Correct Occam3 Constructs in Ada World

Nekkache, M., M. Aliouat and [1]L. Frecon
Département Département Informatique Université Ferhat Abbas
SETIF DZ-19000 Setif Algerie
[1]Laboratoire d'Etude et d'Analyse de la Cognition et des
Modèles Lyon France

**Abstract:** To obtain a broad viewbroad view of parallelism, and fill a gap between wired systems exhibiting natural parallelism and sophisticated theory about high-level language parallel programming, we are building a compiler translating occam3 to Ada95. This study deals with mechanisms elaborated to map correctly Occam3 occam constructs in Ada95 Ada world. This differential study point out effects of those two approaches of parallelism. Trials and comparisons with others parallel systems will bring practical insight about procedures to make parallelism safe effective and usual.

**Key words:** Ada, communication, CSP occam, parallelism, synchronisation, communication, CSP

## INTRODUCTION

Occam is built upon concepts founded by Tony Hoare in CSP and was intended as a low-level assembly language for the Transputer[1-3] and was developed with a minimalist approach that avoids unnecessary duplication of language constructs. occam2 later extended the occam language with the introduction of data types. occam can be used to program a single Transputer or a network of transputers. The occam2 and occam3 programming language enables an application to be described as a collection of concurrent processes. Writing concurrent application is so difficult perhaps we have not the same view like in sequential programming, so we need to have tools for specifying and developing concurrent processes. We suggest that occam seems a good one. In this study we present how occam may be translated to Ada. Similar work was presented by Martin Stuart Mamo for translating occam2 to modula3[4]. Remaki and al[5] present schemata to translating Grafcet to occam2. Occam may then be a bridge for translating Grafcet to Ada which use a derived mechanism of CSP. The Concept of CSP interest more specialists in language development. Branch Hansen extend pascal with procedure call and channel, in super pascal[6].The Southampton portable occam compiler translate occam2 to ANSI C and Peter Welch implement CSP model in JCSP[7]. In an attempt to enlighten analogies and differences, our implementation is done in Ada.

## REPRESENTATION OF CHANNELS AND PROCESS

We are interested only by concurrency of the language at this time. And we present occam under this cornner.

Ooccam is low-level ConcurentConcurrent Programming language[2,8]. Occam is built onto twoonto two fundamental structures: PROCESS, CHANEL. Implementing those structures depend upon disposable environment. Excepting the Transputer, we must choose between languages exhibiting cocurrencyconcurrency, or Sequential Languages interfacing the operating systems. Our implementation is done in Ada95 in an attempt to enlighten analogies and differences.

**Channels:** A channel is a symmetric unbuffered two-pointstwo-point link, allowing two processes to communicate, it may be represented in Ada by an entry as a rendezvous for two communicating tasks[3] an entry is a derived form of DP[9].

**Processes:** In oOccam, the second corner-stone is the process. A processs is an action or a set of actions, executed sequentially or parallel, or by selecting one among several waiting alternatives. The corresponding entity is the Task.

**Synchronization and communication:** synchronization Synchronization and communication in both languages are derived from CSP[8].

**Corresponding Author:** Nekkache, M., Département Département Informatique Université Ferhat Abbas SETIF DZ-19000 Setif Algerie

**Input/Output primitives:** In occam, Input/Output primitives are basic operations realizing both ssynchronization and communication.

**Input from a channel:** For a process P, awaiting a value from channel C is denoted by local command

C?<variable>;

a corresponding Ada construct is given by an ACCEPT on an entry point C for a task P denoted by

accept C(<parameter>);

**Output to channel:** For a process P, sending a value to a process Q via a channel C is denoted by a local Command c! <expression>;

A corresponding Ada construct is given by Q asking a rendezvous on the entry point C of the task P, denoted by P.C(<expression>)

**Process Construct:**

- SEQ Construct correspond to Ada block, each process component correspond to an action.
- PAR construct correspond to an adaAda block, each process component correspond to a task .task.
- ALT construct correspond to a select statement for accept.
- WHILE Construct correspond to while statement
- IF construct correspond to IF endIF ENDIF statement.

**Channel multiplexing:** Occam enables to input a value from one channel among several ones, selecting the first disposable. Such a set of channels may be defined as an array or vector of channels.

In Ada, we may use an array of entries, being awreaware of the following differences. Occam allows sharing of vector of channels by several processes, communicating two by two via a shared element. But Ada assigns a family of entries to one task accepting a rendezvous on oa sole entry at a time.

So Implementing Vector of channels by a family of Entries may be straightforward, but forbids equivalence of parallel schemata.

## PRINCIPLES OF IMPLEMENTATION

**Input/output and channels:** Each element of vector of Channels should correspond to an entry. Each entry is defined in a different task.

Contrary to Ada, occam does not ask for the receiver at an omitted value, the channel being alone named in output command. However, Ada demands might be fulfilled, by associating receiving process to each channel. These associations are built statically by textual analysis. But difficulties arise for multiplexed channels, when a channel is indexedx by a dynamic expression. In such a case , static determination of the used channel(s) is not possible, hence the naming of receiving process. This solution is rejected.

An alternative solution creates a serving task for each element in each vector of channels, as illustrated by the following example.

```
Task type TCANAL is
   Entry LECTURE (Y: in out ELEM);
   Entry ECRITURE (X: in out ELEM)
End TCANAL;
Task body TCANAL is
Begin
  Loop
   Accept
      LECTURE (Y: in out ELEM)
       Do accept
         ECRITURE (X: in out ELEM)
         Do Y: =X End ECRITURE;
       End LECTURE
   End loop;
End TCANAL;
```

For the sake of regularity, this solution is extended to simple channels.

Hence a channel declaration in occam is translataed into a task declaration in Ada. These Tasks will be activated bust before running the body of their englobingenclosing process.

**B processes:** Translating occam processes into ada Ada tasks is straightforward, except for the alternation ALT.

```
ALT
  Guard1
     Guarded process1;
   Guard2
      Guarded process2
…
```

**Examples:**

| occam | Ada |
|---|---|
| CHAN A[10], B | Tache_A:array(0..9) of TCANAL;<br>Tache_B:TCANAL; |
| --receiving process<br>A[exp]?x | Tache_A(exp).LECTURE(x); |
| -- emitting process | Tache_A(exp).ECRITURE(3*2+c); |
| A[exp]! 3*2+c<br>B?y | Tache_B.LECTURE(y); |
| B! result | Tache_B.EcritureECRITURE(result); |

Fig. 1: This rule covers all constructs axceptexcept the alternation ALT

```
occam     Ada
-- exemple                          -- Code
          Task type tache_mere is
            Entry rv_alt1;
           Entry rv_alt2;
Alt        Entry rv_alt3;
     k>5 & B?x                      ....
           proc1                    End tache_mere;
      B?x                           Task body tache_mere is
     Proc2;                         ....
     k=5 & wait t                   Declare
           Proc3                      Tache_alt1:talt_input;alt1;
     Wait  t-1
           Proc4                    Begin
      K=3 &SKIP                       Select
           Proc5                          Wwhen k>5 => =>
.....              accept rrv_alt1_1;
                        Ddo proc1 end;;
          Or  tache_B.LECTURE(x);
                                                        Proc2;
                                    Or when  k=5 =>
                                    accept rv_alt1_2
                Do proc3 end;                                   End
select;
          End;
          .....
          End tache_mere;
          Task type talt 1;
          Task body talt1_input is
          Begin
          select
            Tache_B.LECTURE(x);
                Tache_mere.rv_alt1_1;
          Or     delay t;      Tache_mere.rv_alt1_2
          Or   -- skip avoid     Ttache_mere.rv_alt1_3
          End select;
          End;
```

Fig. 2: Code generated for ALT construct

whereWhere “guard owned 3 conditionnalconditional cases:

conditionCondition & input -- k>5 & B?x
conditionCondition & wait -- i=2 & wait t
conditionCondition and SKIP -- a=2 & SKIP;

Here is an occam example

```
occam
ALT
   k>5 & B?x
      proc1
    k<5 & C?x
       Proc2
```

When we apply the rule Fig. 1 on this example we obtain

```
Select
      When k>5 => tache_B.lecture(x) ;
     Or   when k<5  tache_C.lecture(x)
End select;
```

But when k>5 => tache_B.lecture(x) is an illegal construct in Ada, in, in this context (select accepts).

So a supplementary task is needed to circumvent this restriction. In 1987 we may suggested a task for each alternation[11,12]. But now we propose a better proposal is only one task for the entire ALT construct. Because at any time only one alternation will be satisfied, the select statement warranties that by choosing a unique alternation. That is using a select statement; we decrease the number of tasks generated and then execution time (less task switch).Translation schemata for these conditional alternations Fig. 2.

## CONCLUSION

Translating occam parallelism into Ada parallelism seems very expensive if correct. A reason may be that a micro- parallelism between operations/operators tightly-coupled as in occam has no clear-cut image into the macro-parallelism exhibited by Ada, more oriented towards heavy processes loosely coupled.

## REFERENCES


1. Hoare, C.A.R., 1978. Communicating Sequential Processes, Communication of the ACM.
2. Formal Definition of Occam, 1983. AFCET-informatique, Globule 4, pp: 167-181.
3. Reference manual for the Ada programming language, 1983. United states department Of defence, ANSI/MIL-STD-1815A.
4. Mamo, M.S., 1995. An occam2 To Modula-3 Translator, Diploma In Computer Sci., Magdalene.
5. Remaki, Z., J.F. Ponsignon and M. Nekkache, 1994. Schéma de traduction Grafcet/occam2, third Maghrebian Conference,Engenering on Software and Artificial Intelligence, MCSEAI’94, Rabat 11-14 Avri.
6. Brinch-Hansen, P., 1994. The programming language super Pascal. Software Practice and Experience, 24: 467-484.
7. Welch, P. and J.M.R. Martin, 2000. A CSP model for JAVA thread and vice-versa, Logic and semantic seminar, CU computer Laboratory.
8. Barrett, G., 1992. Occam3 reference manual, Technical report, INMOS Limited, Bristol, BS12 4SQ, England.
9. Hansen, B., 1978. Distributed processes: a concurrent programming concept, communication of the ACM,
10. Hoare, C.A.R., 1984. Programming manual, INMOS limited, Prentice Hill international, series in computer science, C.A.R Hoare series Editor.
11. Nekkache, M., 1987. Système de programmation parallèle occam/Ada, Thèse de Docteur Ingénieur, Insa de Lyon, 17 juille.
12. Nekkache, M., Y. Martinez and L. Frecon, 1987. Expression du parallélisme occam en Ada. Actes des journées ADA AFCET/ENST le Parallélisme en Ada Bigre+globule n°57, Paris decembre , [ISSN 0221-52 ].