INCORPORARE L'ESSENZA DELLA RIFLESSIONE NEL RUN TIME ENVIRONMENT

Maurizio Moro

Seminario IL2

Perchè incorporare l'essenza della riflessione nel RTE?

- coerenza

  riflessione come *funzionalità relativa al sistema*

- convenienza

  costrutti linguistici:

  - low performance
  - meccanismi di notifica al sistema per manipolazioni riflessive

La Riflessione Strutturale in IO

- come si ottiene ?

  completo acceso al layout della memoria dei programmi dei *livelli sottostanti*

- layout della memoria

  quattro segmenti: *segment table*

  code segment

  state and register segment

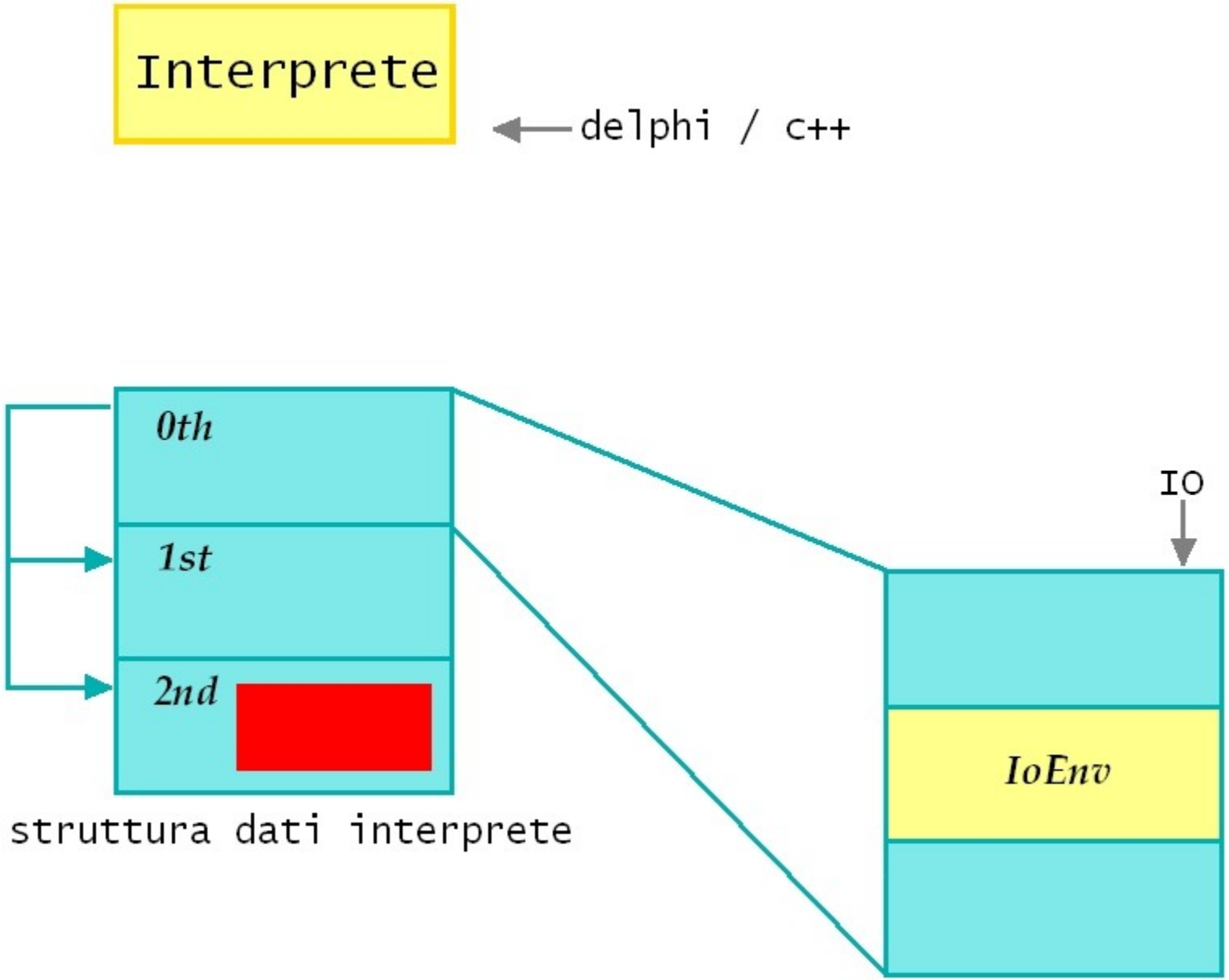
  working storage segment

- virtual storage layout

  Il livello 0 *del display array è riservato per indirizzare i* quattro segmenti del layout di memoria *delle componenti controllate*

Il modulo IoENV

```
MODULE IoEnv(Io_Sys_);
(* This module opens all the low-level data structures of the Io in-
   terpreter implementing the controlled reflective component to the
   importing component. *)

OF Io_Sys USE Io_Word;

GLOBAL StrLng, SgtTop, CodeTop, WsBot, WsTop, pc_, ps_,
       dspy_, sgt_, code_, ir_, ...

CONST StrLng=...; SgtTop=...; CodeTop=...;
      WsBot=...; WsTop=...;
             ...
      TopNest=15; Layers=2; (* number of reflective layers - 1 *)

TYPE Order = RECORD op, a, b: INTEGER END;
     SgtTyp = ARRAY[0..SgtTop] OF SgtElTyp;
     CodTyp = ARRAY[0..CodeTop] OF Order;
     DspTyp = ARRAY[0..TopNest] OF INTEGER;
             ...

VAR sgt_: SgtTyp; (* the segment table *)
    code_: CodTyp; (* the code area *)
    ........
    t_, b_, hb_: INTEGER; (* tos, base and heapbase registers *)
    ........
    ir_: Order;      (* instruction buffer *)
    pc_, ps_: INTEGER; (* program counter and status registers *)
    .......
    dspy_: DspTyp; (* the display *)
    .......
    ws_: ARRAY[WsBot..WsTop] OF Io_Word; (* working storage *)
BEGIN
END.
```


Interprete
delphi / c++
0th
1st
2nd
struttura dati interprete
IO
IoEnv

La Riflessione in IO

La Riflessione in IO è una via di mezzo fra

- *programmazione* "component based"
  - costruzione del programma riflessivo:

    ciascun strato riflessivo è un programma eseguibile in modo indipendente

- *programmazione* "event driven"
  - meccanismo di shift-up e shift-down:

  RTE di IO mappa i meccanismi di shift-up e shift-down riflessivi su un sistema di gestione di interrupt virtuale, emulando l'architettura di un processore reale

Meccanismi di Shifting-Up e di Shifting-Down

- Shift Up

- indirizza il flusso computazionale al meta-livello su impliciti percorsi di esecuzione
- percorsi di esecuzione basati sulla frequenza
- switches di compilazione

- Shift Down

- completamente sotto controllo del programma di meta-livello
- *routine predefinita* RETI(n) forza il flusso di controllo ad uno *shift-down sul livello n*
- *variabile* intlev *del RTE*

Un esempio: Un Debugger Riflessivo

DBX e' attivato prima del programma debuggato, il quale inizia la sua esecuzione dopo la prima chiamata a procedura Shift-Down. Successivamente prende il controllo dopo ciascun breakpoint attivo, o su routine di entrata/uscita, o dopo ciascuna esecuzione di statement, a seconda delle istruzioni in input.

```
(* DBX is a simple line debugger. This module sets/resets breakpoints,
   changes and inspects global values and may execute the controlled
   program by single steps, routines entry/exit or breakpoints. *)

MODULE DBX;

OF DBXDefs USE StrngTyp,pc_,ps_,b_,t_,ShiftDown,PgmDmp,
    MiniDump,WriteTables,SetBreakPt,ResetBreakPt,ShowVar,
    ModVar,InitPgm;

VAR ch:CHAR; radr:INTEGER;id:StrngTyp;

(* dbx input main body *)

BEGIN
  WRITELN('dbx version 1.0');
  WRITELN('ps=',ps_:8,'pc=',pc_:8,'t=',t_:8,'b=',b_:8);
  DO
  WRITE('enter command [b,s,...]>');
  READLN(ch);
  CASE ch OF
    'b','B': (* break *)
        READLN(radr);
        IF radr>0 THEN SetBreakPt(radr)
        ELSE ResetBreakPt(-radr)
        FI\
    'x','X': WRITELN('exiting dbx'); HALT\     (* eXit *)
    'r','R': InitPgm\                          (* Restart *)
    's','S': WRITE('DBX: id>>>'); READLN(id);
             ShowVar(id)\                      (* Show global value *)
    'm','M': WRITE('DBX: id>>>'); READLN(id);
             ModVar(id)\                       (* Modify global value *)
    'c','C','t','T','e','E','y','Y': ShiftDown(ch)
         (* reactivates pgm until next hook or breakpoint *)
    FO;
  OD
END.
```