UDC Release Notes

Peter Grogono and Brian Shearing

28 December 2015

These are release notes for UDC Version "2015 December 28 12:37:18". The updated Desi Report (not quite ready yet, but coming soon) contains detailed descriptions of most of the new or revised features. None of these changes have any effect on the generated DIL code.

1 Major Changes

1.1 Prelude

The Prelude has been revised. The revised version is required by this UDC.

- New syntax for generic routines distinguishes enumerations, maps, protocols, and general types.
- There is a preliminary version of the 'OS' protocols proposed in KIWI 151103.

1.2 Channels and Ports

Assume that *Prot* names a protocol. The declaration p: *Prot* is no longer allowed. The protocol name must be *qualified* in one of the following ways:

chn: *Channel Prot*
cli: *Client Prot*
srv: *Server Prot*

We refer to the qualified entities as 'channels' (*Channel*) and 'ports' (*Client* or *Server*). If p and q and channels or ports, the assignment $p := q$ is allowed if

- the protocols match and
  - either the qualifiers are the same
  - or p is a port and q is a channel.

In other words, channels can be coerced to ports, but not *vice versa*. The same rules apply when channels and ports are passed as arguments.

These changes should have the effect of making Desi more secure and expressive than previously. Processes can communicate only with ports that have defined directions, and the messages sent are guaranteed to be compatible with protocols.

Protocol matching is still under development and, in this respect, there are no new advances in the UDC.

1.3 'new' Statement

The '**new**' statement creates a new component of a map of channels or ports.

- Syntax: **new** $m[i]$. '**new**' is a keyword.
- Effect: create a new component of map m at location i.
- The operand must be a subscripted expression.
- The map must have a range type that is a channel or port.

Example: the Desi program

Tran = **protocol** { *start*; *stop* }
Map = **type** *Word* **indexes** *Channel Tran*;
Proc = **process** {
    m: *Map*;
    new *m*[0]
}

generates DIL code that includes

```
blk
  var U[WP]1674 "m"
  tmp GP1704
edcl
  line LW7
  mcpy GP1704 U[WP]1674 LW0
eblk
```

The restriction to arguments that are channels or ports could easily be removed if it turns out to be useful to employ '**new**' with other types.

1.4 Nested Selectors

Expressions of the form $p.q.f$ are supported. The dot operator associates to the left: $p.q.f$ is interpreted as $(p.q).f$,[1] with $p.q$ yielding a port that is then applied to the protocol field f.

For example, the Desi program

Trans = **protocol** { *data*: *Word* }
Nest = **protocol** { *chan*: *Client Trans* }
Client = **process** *pn*: +*Nest* {
    pn.chan.data := 32
}

generates DIL code that includes

```
chn UP1673 LW1 "Server pn"
  fld LW0 "Client chan"
blk
```

[1] However, Desi syntax does not allow the parentheses.

```
   tmp GP1705
   tmp GW1709
edec
   rcv GP1705 UP1673 LW0
   cpy GW1709 LW32
   snd GW1709 GP1705 LW0
eblk
```

2 Minor Modifications

2.1 `I` Option

The UDC options '`+In`' sets indentation for DIL code to n spaces.

- The default (no `I` option) is $n = 0$ (no indentation).
- '`-In`' has the same effect as '`+In`'.
- Label commands '`lbl`' are not indented.

The default setting minimizes the size of DIL files. Indentation is intended for situations in which human-readable DIL is needed.

2.2 Map types

UDC rejects a map declaration if the domain type of the map is any of:

- *Real*;
- **mod** r with r: *Real*;
- a protocol;
- a map type.

3 Towards I/O Primitives

KIWI 151103, "Really primitive I/O Primitives", suggested that operating system services should be provided by means of protocols rather than libraries of routines. Programs included in this release take a step in this direction.

The program `test-os` follows the proposed convention: it defines a cell *Main* with a port parameter but does not invoke any cells or processes,

```
Main = process os: Client OS
{
    scr: Client Screen := os.scr
    kbd: Server Keyboard := os.kbd
    scr.out := 'Enter name: '
    name: Text := kbd.line
    scr.out := 'Hallo, ' // name // '!'
}
```

```
import 'osim';
```

The magic that allows `test-os` to execute has two parts. First, the Prelude contains the required protocol definitions. Currently, we provide only screen and keyboard interfaces; others will be added later.

```
Screen = protocol { *(out: Text | outln: Text) }
Keyboard = protocol { *(char: Text | line: Text) }
OS = protocol {*(↑scr: Client Screen | ↑kbd: Server Keyboard) }
```

Second, the **import** directive in `test-os` imports code that simulates the behaviour of the operating system services by using the existing library routines. When this code (or its equivalent) has been built into the run-time system, the library routines and **import** will no longer be needed.

Some of the guarded sequences could be reduced to single assignments but have been left in expanded form for clarity.

```
Simulator = process os: Server OS
{
    scrChan: Channel Screen
    scrPort: Server Screen :=scrChan
    kbdChan: Channel Keyboard
    kbdPort: Client Keyboard := kbdChan
    loop select
    {
        || os.scr := scrChan
        || os.kbd := kbdChan
        || msg: Text := scrPort.out;
           scrch(msg)
        || msg: Text := scrPort.outln;
           scrln(msg)
        || msg: Text := kbdch();
           kbdPort.char := msg
        || msg: Text := kbdln();
           kbdPort.line := msg
    }
}

System = cell
{
    chn: Channel OS
    Main(chn)
    Simulator(chn)
}

System()
```