Translation of DOL Application Specification to BIP

Matthieu Gallien

VERIMAG

March 27, 2009

Introduction

- Joint work with Iuliana Bacivarov, Wolfgang Haid, Kai Huang and Lothar Thiele from ETHZ-TIK
- DOL is a specification framework for dataflow embedded systems:
    - Specifications for the Application, Hardware and Mapping
    - Behavior of the Application is C/C++ code
- Translation of DOL Application Specification without taking into account the hardware
    - Automatic translation of DOL C/C++ behavior code to BIP behavior model
        - Translation based on a model of C/C++ code
    - A model of the software independently of the platform

Outline

1. Translating a DOL Application to a BIP Model
   - Quick Summary of DOL
   - Model of DOL Application Specification
   - BIP Models for C Code Elements
   - Implementation

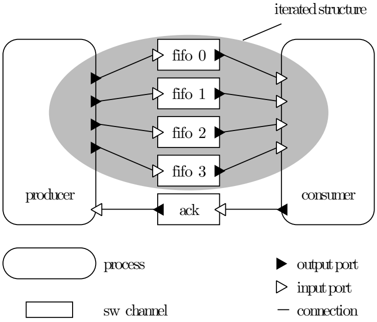
2. Examples

3. Conclusion

DOL is a Specification Framework for Data Flow Embedded Systems

- A specification for the application
  - 3 basic entities: process, software channel and connection
  - All processes have the same organisation: an `init` procedure then a continuous loop of the `fire` procedure
  - The behavior of procedures `init` and `fire` is described by C language with some added constraints:
    - There is some special functions: `DOL_write` and `DOL_read` for the data transfers

Example of DOL process



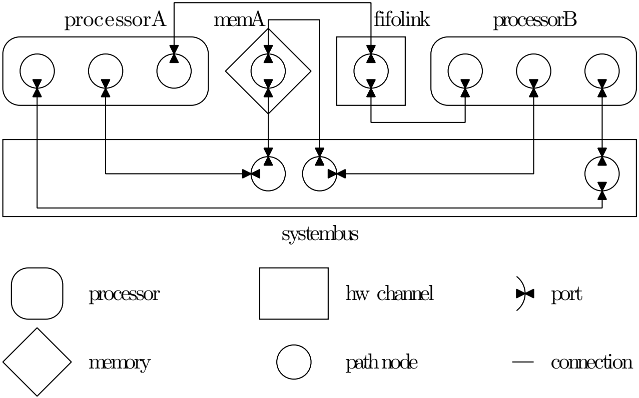
Process Model

```
procedure INIT(DOLProcess $p$)                      initialization
    initialize local data structures
end procedure

procedure FIRE(DOLProcess $p$)                      execution
    DOL_read(INPUT, size, buf)                      blocking read
    manipulate
    DOL_write(OUTPUT, size, buf)                    blocking write
end procedure
```

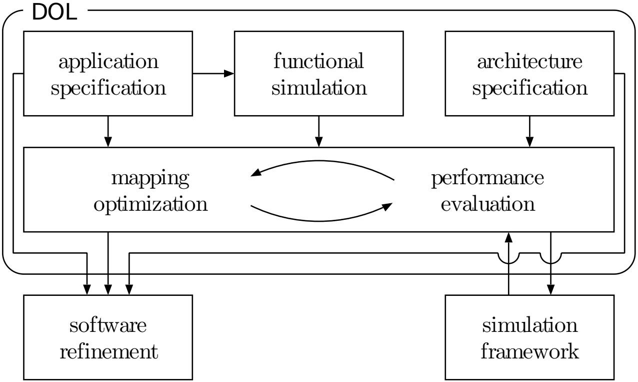
DOL is a Specification Framework for Data Flow Embedded Systems

- A specification of hardware architecture
  - Includes all processors, memories, buses and possible connections between buses and memories
  - Each hardware element has a type corresponding to a real hardware element
  - Hardware elements can be parameterized
- A specification of the mapping of an application on the hardware architecture
  - Includes the mapping of processes to processors and of software channels to hardware channels
  - Includes schedules for processors and buses

Example of DOL Hardware Architecture



DOL Workflow



DOL can Optimize the Load of Hardware Architecture

- From the results of:
    - A purely functional simulation: does not take into account the hardware architecture
    - An instruction accurate simulation: takes into account the hardware architecture
- From the performance analysis, an optimization of the mapping is done with genetic algorithms

A DOL Application Specification

$$\textbf{Application} ::= (\textbf{Process})^{+}\ (\textbf{SWChannel})^{+}\ (\textbf{Connection})^{+}$$

$$\textbf{Process} ::= (\mathit{ProcInPort} + \mathit{ProcOutPort})^{+}\ \textbf{Behavior}$$

$$\textbf{SWChannel} ::= \mathit{Size}\ \mathit{RecvPort}\ \mathit{SendPort}$$

$$\textbf{Connection} ::= (\mathit{ProcOutPort}\ \mathit{RecvPort}) + (\mathit{SendPort}\ \mathit{ProcInPort})$$

Summary of the C Code Model

$$\begin{aligned}
\textbf{Behavior} &::= \textbf{procedure procedure} \\[1em]
\textbf{procedure} &::= (\textit{variable\_declaration})^{*}\ \textbf{statement\_group} \\[1em]
\textbf{statement\_group} &::= (\textbf{for} + \textbf{if} + \textbf{switch} + \textit{DOL\_read} + \\
&\quad\ \textit{DOL\_write} + \textit{simple\_statement} + \\
&\quad\ \textbf{statement\_group})^{+} \\[1em]
\textbf{if} &::= \textit{condition}\ \textbf{statement\_group} \\
&\quad\ \left[\textbf{statement\_group}\right]
\end{aligned}$$

Summary of the C Code Model

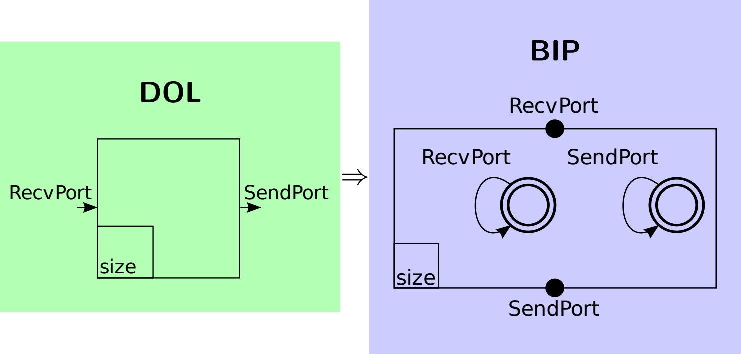
$$\begin{aligned}
\textbf{for\_statement\_group} &::= \Big(\textit{simple\_statement} + \\
&\qquad \textbf{statement\_group}\Big) * \\
\\
\textbf{for} &::= \textbf{for\_statement\_group}\ \textit{condition} \\
&\qquad \textbf{for\_statement\_group} \\
&\qquad \textbf{statement\_group} \\
\\
\textbf{switch} &::= \textit{variable}\ (\textbf{case})^{+}\ \Big[\textbf{default}\Big] \\
\textbf{case} &::= \textit{value}\ \textbf{statement\_group} \\
\textbf{default} &::= \textit{default}\ \textbf{statement\_group}
\end{aligned}$$

Restriction of C/C++ Code Accepted in DOL Behavior

- All variable declarations should be at the beginning of `init` and `fire` procedures
- Only one `return` by procedure
- No `goto`
- All `DOL_read`, `DOL_write`, `DOL_detach` in the init and fire procedure: not in external functions called in those procedure

Example of Translation of DOL software channel

SWChannel ::= *Size RecvPort SendPort*



Translation of DOL connections

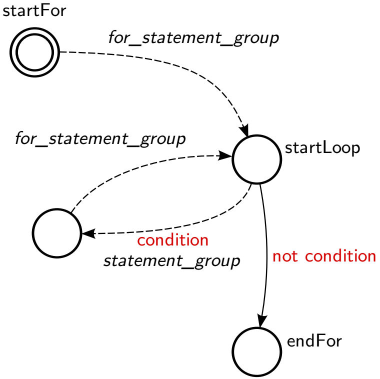
$\textbf{Connection}_{DOL}$ ::= *output input*

$\textbf{Connection}_{BIP}$ ::= connector(*output*, *input*)

- For each connection corresponds one BIP connector
  - It will transfer the size and the address of the data to be transfered;
  - It will also synchronise the two components during the copy of the data.

For

for ::= **for_statement_group** *condition* **statement_group for_statement_group**

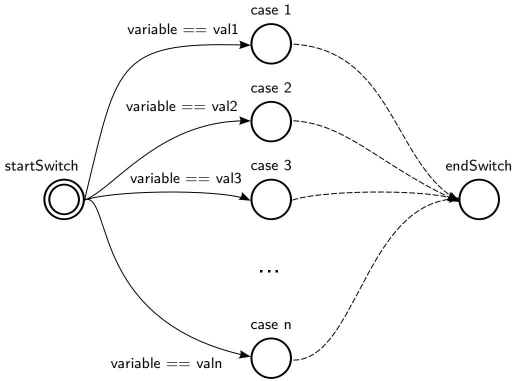


Switch

$$\textbf{switch} ::= \textit{variable}.(\textbf{case})^{+}.\left[\textbf{default}\right]$$

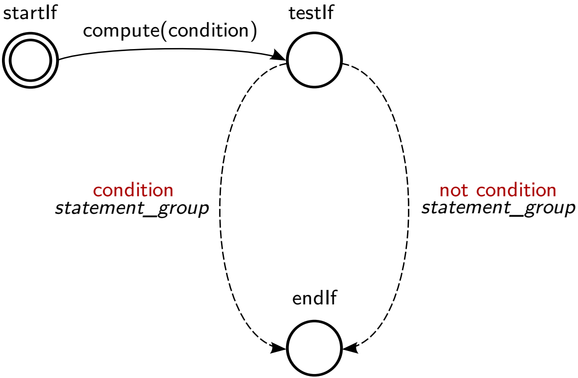
$$\textbf{case} ::= \textit{value}.\textbf{statement\_group}$$

$$\textbf{default} ::= \textit{default}.\textbf{statement\_group}$$



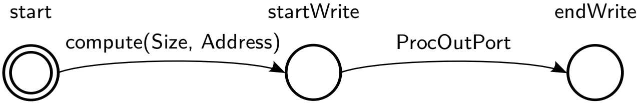
If

if ::= *condition* **statement_group** [**statement_group**]

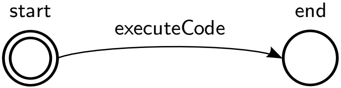


DOL_write

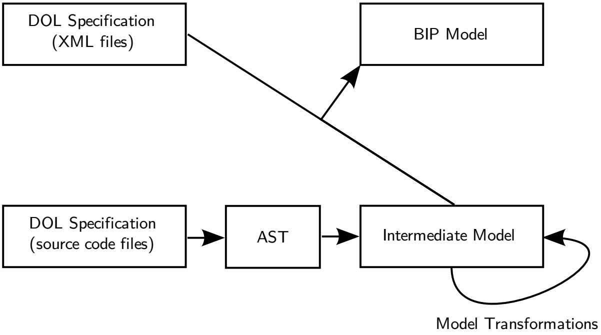
DOL_write ::= *ProcOutPort Size Address*



Simple Statement



Compiler Architecture


DOL Specification (XML files)
BIP Model
DOL Specification (source code files)
AST
Intermediate Model
Model Transformations

Model Transformations

1. Transform code that is not conformant to the restrictions we put on C code into conformant code
2. Generate code for initializing variables
3. Replace all DOL specificities into something equivalent but with no dependencies to DOL (access to internal data of DOL implementation, ...)

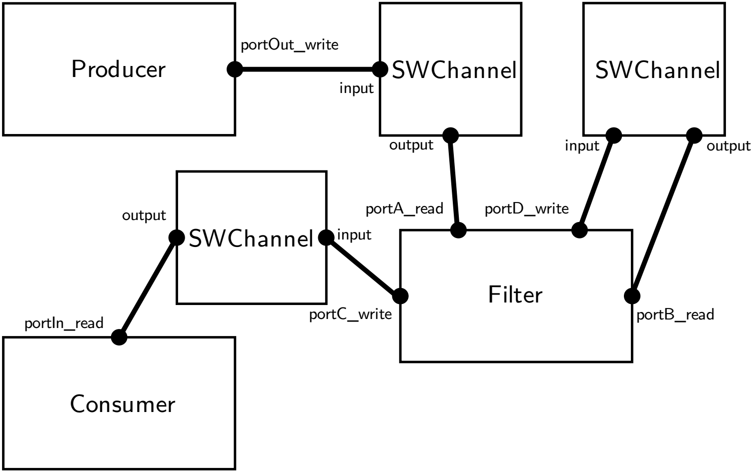
Compiler Optimisation

1. Collapsing code tree without DOL special functions

2. Merging sequential code without DOL special functions

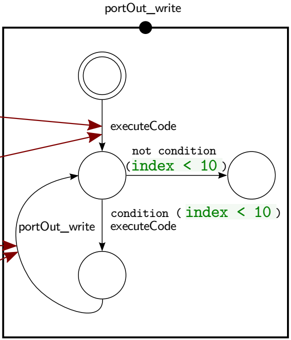
3. Transforming `for` loops into simpler `while` loops if possible (i.e. does not contain `continue` statements)

Complete Filter Example

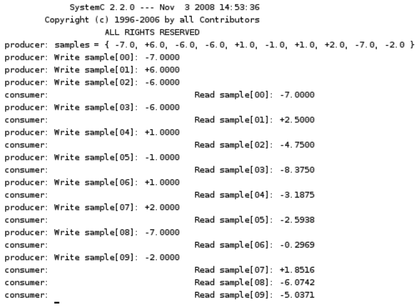


Filter Example: Producer Process

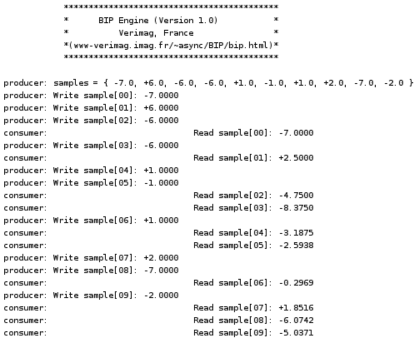
```
int producer_fire(DOLProcess *p)
{
  int index;

  srand(0); //initialize random number generator

  //generate input samples and display them
  printf("producer: samples = { ");

  for (index = 0; index < 10; index++) {
    p->local->sample[index] = (float) getRandomNumber(-9, 9);
    if (index < 9) {
      printf("%+3.1f, ", p->local->sample[index]);
    }
    else {
      printf("%+3.1f }\n", p->local->sample[index]);
    }
  }

  //write samples to output port
  for (index = 0; index < 10 ; index++) {
    printf("%8s: Write sample[%02d]: %+6.4f\n",
           "producer", index, p->local->sample[index]);
    DOL_write((void*)PORT_OUT, &(p->local->sample[index]),
              sizeof(float), p);
  }

  DOL_detach(p);
  return -1;
}
```



Trace of Filter Example



```
             SystemC 2.2.0 --- Nov  3 2008 14:53:36
        Copyright (c) 1996-2006 by all Contributors
                    ALL RIGHTS RESERVED
producer: samples = { -7.0, +6.0, -6.0, -6.0, +1.0, -1.0, +1.0, +2.0, -7.0, -2.0 }
producer: Write sample[00]: -7.0000
producer: Write sample[01]: +6.0000
producer: Write sample[02]: -6.0000
consumer:                             Read sample[00]: -7.0000
producer: Write sample[03]: -6.0000
consumer:                             Read sample[01]: +2.5000
producer: Write sample[04]: +1.0000
consumer:                             Read sample[02]: -4.7500
producer: Write sample[05]: -1.0000
consumer:                             Read sample[03]: -8.3750
producer: Write sample[06]: +1.0000
consumer:                             Read sample[04]: -3.1875
producer: Write sample[07]: +2.0000
consumer:                             Read sample[05]: -2.5938
producer: Write sample[08]: -7.0000
consumer:                             Read sample[06]: -0.2969
producer: Write sample[09]: -2.0000
consumer:                             Read sample[07]: +1.8516
consumer:                             Read sample[08]: -6.0742
consumer:                             Read sample[09]: -5.0371
```



```
            *******************************************
            *      BIP Engine (Version 1.0)           *
            *          Verimag, France                *
            *(www-verimag.imag.fr/~async/BIP/bip.html)*
            *******************************************
producer: samples = { -7.0, +6.0, -6.0, -6.0, +1.0, -1.0, +1.0, +2.0, -7.0, -2.0 }
producer: Write sample[00]: -7.0000
producer: Write sample[01]: +6.0000
producer: Write sample[02]: -6.0000
consumer:                             Read sample[00]: -7.0000
producer: Write sample[03]: -6.0000
consumer:                             Read sample[01]: +2.5000
producer: Write sample[04]: +1.0000
producer: Write sample[05]: -1.0000
consumer:                             Read sample[02]: -4.7500
consumer:                             Read sample[03]: -8.3750
producer: Write sample[06]: +1.0000
consumer:                             Read sample[04]: -3.1875
consumer:                             Read sample[05]: -2.5938
producer: Write sample[07]: +2.0000
producer: Write sample[08]: -7.0000
consumer:                             Read sample[06]: -0.2969
producer: Write sample[09]: -2.0000
consumer:                             Read sample[07]: +1.8516
consumer:                             Read sample[08]: -6.0742
consumer:                             Read sample[09]: -5.0371
```

Conclusion and Future Work

- Fully automatic translation of 7 DOL examples
- Produces a BIP model that is usable for model transformations
- Need to support more complex examples like MPEG decoder
- We want to implement model transformations in order to apply hardware constraints to the model of the software