An Effect Type System for Modular Distribution of Dataflow Programs

Gwenaël Delaval[1] Alain Girault[1] Marc Pouzet[2]

[1] INRIA Rhône-Alpes, Pop-Art project

[2] LRI, Demons team

June 13, 2008 — LCTES, Tucson, Arizona

Motivations

- To provide a language-oriented solution for the design of functionally distributed systems:

Motivations

- To provide a language-oriented solution for the design of functionally distributed systems:

  Alternative: separate design of each computing resource.
  Which raises the following problems:
  - One function can involve several computing resources
    $\Longrightarrow$ separate design of closely related components, risks of data inconsistency.
  - One computing resource can be involved in several functions
    $\Longrightarrow$ duplicated control jeopardizing the modularity.

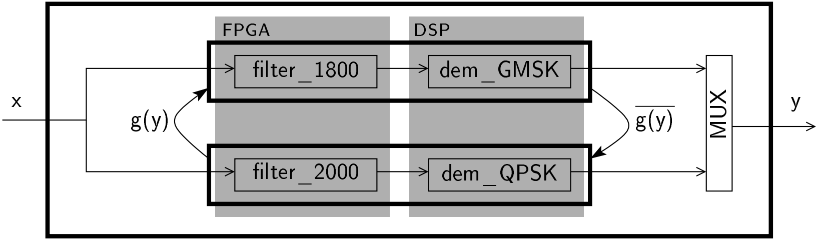
Motivations

- To provide a language-oriented solution for the design of functionally distributed systems:

  Alternative: separate design of each computing resource. Which raises the following problems:
  - One function can involve several computing resources $\Longrightarrow$ separate design of closely related components, risks of data inconsistency.
  - One computing resource can be involved in several functions $\Longrightarrow$ duplicated control jeopardizing the modularity.
- To perform modular distribution:
  - To avoid inlining every function
  - To allow, by mean of high-order features, dynamic reconfiguration of a resource by another (by sending functions through channels)

Example

Multichannel reception system of a software-defined radio



Reception channel composed of two components:

- a pass-band filter implemented on a FPGA
- a demodulator implemented on a DSP

Synchronous Reactive Systems

Reactive systems

A reactive system is a system which reacts to its environment at the speed of the environment

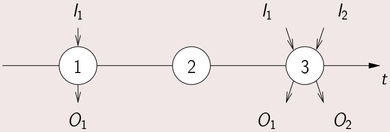
Synchronous Reactive Systems

Reactive systems

A reactive system is a system which reacts to its environment at the speed of the environment

The Synchronous Hypothesis

- Discrete time scale



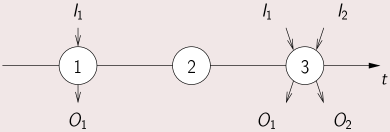
Synchronous Reactive Systems

Reactive systems

A reactive system is a system which reacts to its environment at the speed of the environment

The Synchronous Hypothesis

- Discrete time scale



- Instantaneous broadcast of events

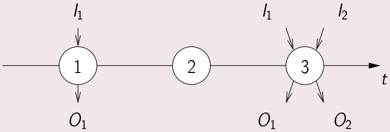
Synchronous Reactive Systems

Reactive systems

A reactive system is a system which reacts to its environment at the speed of the environment

The Synchronous Hypothesis

- Discrete time scale



- Instantaneous broadcast of events
- Language restrictions to ensure real-time executions

Dataflow synchronous languages

Example

```
node sum (x,reset) = s where
  s = x + (0 -> if reset then 0
                else pre s)
val sum :  int * int -> int
val sum :: 'a * 'a -> 'a
```

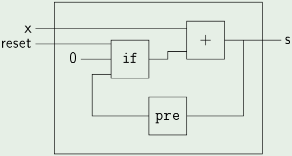
- Examples: Lustre, Lucid Synchrone
- `x`, `reset` and `s` are *data streams*
- state of the system represented by the *memories* (`pre`(e))

Dataflow synchronous languages

Example

```
node sum (x,reset) = s where
  s = x + (0 -> if reset then 0
                else pre s)
val sum :  int * int -> int
val sum :: 'a * 'a -> 'a
```

$$\equiv$$



- Examples: Lustre, Lucid Synchrone
- `x`, `reset` and `s` are *data streams*
- state of the system represented by the *memories* (pre(e))

SDR example

```
node channel(filter,demod,x) = y where
      m = filter(x)
  and y = demod(m)

node multichannel_sdr(x) = y where
      c = g(y)
  and match (true fby c) with
      | true -> y = channel(filter_bp_1800, demod_gmsk, x)
      | false -> y = channel(filter_bp_2000, demod_qpsk, x)
```

SDR example

```
loc FPGA; loc DSP; loc GPP;
link FPGA to DSP; link DSP to GPP;

node channel(filter,demod,x) = y where
      m = filter(x) at FPGA
  and y = demod(m) at DSP

node multichannel_sdr(x) = y where
      c = g(y)
  and match (true fby c) with
      | true -> y = channel(filter_bp_1800, demod_gmsk, x)
      | false -> y = channel(filter_bp_2000, demod_qpsk, x)
```

Distribution of synchronous dataflows programs

Principles

- Allowing the explicit localization of computations:
  `y = f(x) at A ...`
- Inference of the localization of each value and computation from expressed ones: "coloration" of the program [Caspi, Girault & Pilaud, IEEE TSE 1999]

Distribution of synchronous dataflows programs

Principles

- Allowing the explicit localization of computations:
  `y = f(x) at A ...`
- Inference of the localization of each value and computation from expressed ones: “coloration” of the program [Caspi, Girault & Pilaud, IEEE TSE 1999]

Realisation

- Localization of a value ↔ spatial type of this value
- “coloration” ↔ type inference

Extended language

- Architecture description : composed of location declarations as symbolic names, and links between locations :

```
loc FPGA;
loc DSP;
loc GPP;
link FPGA to DSP;
link DSP to GPP;
```

- Localization : e at A : every computation in e will be performed at location A
- Communications : abstraction of their positions, and their technical expression

Annotated semantics : principle

Based on a reactive semantics : $R \vdash e \xrightarrow{v} e'$.

- Formalization of distributed values $\hat{v}$ (and distributed reaction environment $\hat{R}$) :
  - i `at` s : immediate value located at s
  - (vl, vl') `at` A : pair entirely located at A
  - $(vl$ `at` A, vl' `at` $B)$: distributed pair
- Formalization of distributed reactions involving a set of sites ℓ : $\hat{R} \overset{\ell}{\Vdash} e \xrightarrow{\hat{v}} e'$

Distributed execution I

- An immediate value can be emitted anywhere :

$$\text{(Imm)}\quad \hat{R} \overset{\{s\}}{\Vdash} i \xrightarrow{i\ \texttt{at}\ s} i$$

- An operation can be performed only on two values at the same location, and the result is on this location :

$$\text{(Op)}\quad \frac{\hat{R} \overset{\ell_1}{\Vdash} e_1 \xrightarrow{i_1\ \texttt{at}\ A} e_1' \qquad \hat{R} \overset{\ell_2}{\Vdash} e_2 \xrightarrow{i_2\ \texttt{at}\ A} e_2' \qquad i = \mathrm{op}(i_1, i_2)}{\hat{R} \overset{\ell_1 \cup \ell_2}{\Vdash} \mathrm{op}(e_1, e_2) \xrightarrow{i\ \texttt{at}\ A} \mathrm{op}(e_1', e_2')}$$

Distributed execution II

- The reaction of a `e at A` expression must involve at most the location `A` :

(At)

$$\frac{\hat{R} \overset{\{A\}}{\Vdash} e \xrightarrow{\hat{v}} e'}{\hat{R} \overset{\{A\}}{\Vdash} e \ \mathtt{at}\ A \xrightarrow{\hat{v}} e' \ \mathtt{at}\ A}$$

- A value located on one site `A` can be communicated on any location available from `A` :

(Comm)

$$\frac{\hat{R} \overset{\ell}{\Vdash} e \xrightarrow{vl\ \mathtt{at}\ A} e' \qquad (A, A') \in \mathcal{L}}{\hat{R} \overset{\ell \cup \{A'\}}{\Vdash} e \xrightarrow{vl\ \mathtt{at}\ A'} e'}$$

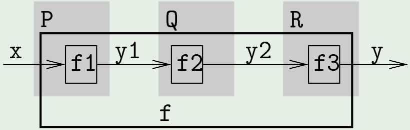
Spatial types: overview

Type and effect system [Talpin & Jouvelot, JFP 1992] :

- t at s is the spacial type of a value available at location s (either a constant location A or a location variable δ)
- functional values: spatial types of the form $s_i \prec\!\langle \ell, T \rangle\!\rightarrowtail s_o$ s.t. :
  - s_i is the spatial type of the function's inputs
  - s_o is the spatial type of the function's outputs
  - ℓ is the set of locations involved in the computation of the function (used for type inference: it includes every location of s_i, s_o, and T)
  - T is a set of communication channels involved in the computation (used for distribution)

Example of spatial types

```
node f(x) = y3 where
        y1 = f1(x) at P
  and y2 = f2(y1) at Q
  and y3 = f3(y2) at R
```



Assuming that `f1`, `f2`, and `f3` are computable everywhere :

- `f1` gets the spatial type $c \;\mathtt{at}\; \mathtt{P} -\!\!\langle \{\mathtt{P}\}, \emptyset \rangle\!\!\rightarrow c \;\mathtt{at}\; \mathtt{P}$
- `f2` gets the spatial type $c \;\mathtt{at}\; \mathtt{Q} -\!\!\langle \{\mathtt{Q}\}, \emptyset \rangle\!\!\rightarrow c \;\mathtt{at}\; \mathtt{Q}$
- `f3` gets the spatial type $c \;\mathtt{at}\; \mathtt{R} -\!\!\langle \{\mathtt{R}\}, \emptyset \rangle\!\!\rightarrow c \;\mathtt{at}\; \mathtt{R}$
- because of the presence of the two communications, `f` is finally of spatial type $c \;\mathtt{at}\; \mathtt{P} -\!\!\langle \{\mathtt{P}, \mathtt{Q}, \mathtt{R}\}, [\mathtt{P} \overset{1}{\mapsto} \mathtt{Q}, \mathtt{Q} \overset{2}{\mapsto} \mathtt{R}] \rangle\!\!\rightarrow c \;\mathtt{at}\; \mathtt{R}$

Spatial type expressions

Judgments of the form $H \vdash e : t/\ell/T$:

In the environment H, the expression e is of spatial type t, and computing e involves :

- the set of locations ℓ
- the set of named communication channels T

Typing rules: localization

The expression e at A is typed by constraining the computation of the expression e to involve at most the location A, by use of instanciation mechanisms :

(At)

$$\frac{H \texttt{ at } A \vdash e : t/\{A\}/T \qquad A \in \mathcal{S}}{H \vdash e \texttt{ at } A : t/\{A\}/T}$$

Typing rules: communications

Communications are inserted by means of subtyping :

A value typed t at s can be considered as of type t at s', provided that there exists a communication link from s to s' :

$$\text{(Comm)}\quad \frac{H \vdash e : tc \text{ at } s/\ell/T \qquad (s, s') \in \mathcal{L}}{H \vdash e : tc \text{ at } s'/\ell \cup \{s'\}/T, [s \overset{c}{\hookrightarrow} s']}$$

Distribution: Principle

- Definition of a type-directed operation of projection :

$$H \vdash D : H'/\ell/T \xrightarrow{A} D'$$

$$H \vdash e : t/\ell/T \xrightarrow{A} e'/D$$

The expression e, projected on A, results in a new expression e'

D contains additional declarations (e.g., channel definitions)

Distribution: Principle

- Definition of a type-directed operation of projection :

$$H \vdash D : H'/\ell/T \xRightarrow{A} D'$$

$$H \vdash e : t/\ell/T \xRightarrow{A} e'/D$$

The expression e, projected on A, results in a new expression e'

D contains additional declarations (e.g., channel definitions)

- Communication channels used between two projected expressions are added as inputs or outputs streams of nodes

Modular distribution: principle

Channel names, defined or used as streams in the body of a function, will be added to the signature of this function

Modular distribution: principle

Channel names, defined or used as streams in the body of a function, will be added to the signature of this function

This allows multiple instanciations of functions

Distribution example: node distribution

```
loc A;
loc B;
link A to B;
node f(x) = z where
       y = x + 1 at A
   and z = y + 2 at B
```

Distribution example: node distribution

```
loc A;
loc B;
link A to B;
node f(x) = z where
      y = x + 1 at A
  and z = y + 2 at B
```

Spatial type of `f` : $b\ \mathtt{at}\ A \ {-\!\!\langle} \{A, B\} / [A \overset{c}{\mapsto} B] {\rangle\!\!\rightarrow}\ b\ \mathtt{at}\ B$

Distribution example: node distribution

```
loc A;
loc B;
link A to B;
node f(x) = z where
        y = x + 1 at A
   and z = y + 2 at B
```

Spatial type of `f` : $b \text{ at } A \prec\!\langle\{A, B\}/[A \overset{c}{\mapsto} B]\rangle\!\succ b \text{ at } B$

⟶ On A, c is added as an output :

```
node f(x) = ((),c) where y = x + 1 and c = y
```

Distribution example: node distribution

```
loc A;
loc B;
link A to B;
node f(x) = z where
        y = x + 1 at A
   and z = y + 2 at B
```

Spatial type of `f` : $b \text{ at } A \multimap\!\langle\{A, B\}/[A \overset{c}{\mapsto} B]\rangle\!\!\longrightarrow b \text{ at } B$

⟶ On A, c is added as an output :

```
node f(x) = ((),c) where y = x + 1 and c = y
```

⟶ On B, c is added as an input :

```
node f (x,c) = z where z = c + 2
```

Distribution example: node instanciation

```
node g (x1,x2) = (y1,y2) where
     y1 = f(x1)
 and y2 = f(x2)
```

$$\mathrm{g} : (b \text{ at } A \times b \text{ at } A) \;{-\!\!\langle}\, \{A, B\} / [A \overset{c_1}{\mapsto} B, A \overset{c_2}{\mapsto} B] \,{\rangle\!\!\rightarrow}\; (b \text{ at } B \times b \text{ at } B)$$

Distribution example: node instanciation

```
node g (x1,x2) = (y1,y2) where
     y1 = f(x1)
 and y2 = f(x2)
```

$$\mathrm{g} : (b \text{ at } A \times b \text{ at } A) \;-\!\!\langle\{A, B\}/[A \overset{c_1}{\hookrightarrow} B, A \overset{c_2}{\hookrightarrow} B]\rangle\!\!\rightarrow\; (b \text{ at } B \times b \text{ at } B)$$

⇒ use of two distinct channels : one for each instanciation

Node instanciation: projection on A and B

On location A :

- Channel c_1 is an output of `f`'s first instance (resp. c_2)
- Channels c_1 and c_2 are both added as outputs of `g`

```
node g (x1,x2) = ((x1,x2),c1,c2) where
     (x1,c1) = f(x1)
 and (x2,c2) = f(x2)
```

Node instanciation: projection on A and B

On location A :

- Channel c_1 is an output of `f`'s first instance (resp. c_2)
- Channels c_1 and c_2 are both added as outputs of g

```
node g (x1,x2) = ((x1,x2),c1,c2) where
     (x1,c1) = f(x1)
 and (x2,c2) = f(x2)
```

On location B :

- Channel c_1 is an input of `f`'s first instance (resp. c_2)
- Channels c_1 and c_2 are both inputs of g

```
node g ((x1,x2),c1,c2) = (y1,y2) where
     y1 = f(x1,c1)
 and y2 = f(x2,c2)
```

SDR example

```
node channel(filter,demod,x) = y where
     m = filter(x) at FPGA
 and y = demod(m) at DSP
```

$$\begin{aligned}\texttt{channel} : \forall \alpha,\beta,\gamma,\eta.&(\alpha\ \texttt{at FPGA} \mathrel{-\!\!\langle\{\texttt{FPGA}\}/\emptyset\rangle\!\!\rightarrow} \beta\ \texttt{at FPGA})\\ &\times (\beta\ \texttt{at DSP} \mathrel{-\!\!\langle\{\texttt{DSP}\}/\emptyset\rangle\!\!\rightarrow} \gamma\ \texttt{at DSP})\\ &\times \alpha\ \texttt{at FPGA}\\ &\mathrel{-\!\!\langle\{\texttt{FPGA},\texttt{DSP}\}/[\texttt{FPGA} \overset{c}{\hookrightarrow} \texttt{DSP}]\rangle\!\!\rightarrow} \gamma\ \texttt{at DSP}\end{aligned}$$

Control: SDR example

```
node multichannel_sdr(x) = y where
     c = g(y)
 and match (true fby c) with
  | true -> y = channel(filter_bp_1800, demod_GMSK, x)
  | false -> y = channel(filter_bp_2000, demod_QPSK, x)
```

$$\texttt{multichannel\_sdr} : c\ \texttt{at}\ \texttt{FPGA} \,-\!\!\langle \{\texttt{FPGA}, \texttt{DSP}\} / T \rangle\!\!\rightarrow c\ \texttt{at}\ \texttt{DSP}$$

with $T = [\texttt{FPGA} \overset{c_1}{\mapsto} \texttt{DSP}, \texttt{FPGA} \overset{c_2}{\mapsto} \texttt{DSP}, \texttt{DSP} \overset{c_3}{\mapsto} \texttt{FPGA}]$

Results

Theorems

- Executable by centralized semantics $\wedge$ typable $\Rightarrow$ executable by the distributed semantics

Results

Theorems

- Executable by centralized semantics $\wedge$ typable $\Rightarrow$ executable by the distributed semantics
- Let D distributed on an architecture $G = \langle A_1, \dots, A_n, \mathcal{L} \rangle$:

$$H \vdash D : H' / \ell / T \xRightarrow{A_1} D_1$$
$$\vdots$$
$$H \vdash D : H' / \ell / T \xRightarrow{A_n} D_n$$

Then D is semantically equivalent with D_1 `and` ... `and` D_n.

Results

Theorems

- Executable by centralized semantics $\wedge$ typable $\Rightarrow$ executable by the distributed semantics
- Let D distributed on an architecture $G = \langle A_1, \ldots, A_n, \mathcal{L} \rangle$:

$$H \vdash D : H' / \ell / T \overset{A_1}{\Longrightarrow} D_1$$
$$\vdots$$
$$H \vdash D : H' / \ell / T \overset{A_n}{\Longrightarrow} D_n$$

Then D is semantically equivalent with D_1 `and` ... `and` D_n.

Implementation

Type system with channel inference and automatic distribution implemented into the Lucid Synchrone compiler.

Contributions

- Interest of a language approach for the design of distributed embedded systems
- Extension of a synchronous dataflow language with primitives for program distribution
- Proposal of a formal semantics for this extended language, through the formalization of distributed values and distributed execution
- Design of a spatial type system, and a type-directed automatic distribution operation : integration within modular compilation
- Taking into account of control primitives : global clocks and `match`/`with`
- Implementation into the Lucid Synchrone compiler

Prospects

- Finer architecture description: typed channels, hierarchy, clock or temporal informations
- Finer and more complete distribution via the expression of more general types, including site or effect variables
- Examination of higher-order dataflow programs distribution. Goal: expression of dynamic reconfiguration of a resource by sending functions through channels. . .

Prospects: dynamic reconfiguration

This is 100% valid Lucid Synchrone code !!!

```
let node channel reconfigure x = y where
  rec automaton
      | Init ->
          do y = x
          until reconfigure(filter,demod)
          then Configure(filter,demod)
      | Configure(filter,demod) ->
          let f = run filter x in
          do y = run demod f
          until reconfigure(filter',demod')
          then Configure(filter',demod')
      end
```

Prospects: dynamic reconfiguration (cont'd)

```
let node multichannel_sdr x = y where
  rec y = channel switch_channel x
  and automaton
      | Reconfigure_GSM ->
          do emit switch_channel = (filter_1800,demod_gmsk)
          then GSM
      | GSM ->
          do until (umts y) then Reconfigure_UMTS
      | Reconfigure_UMTS ->
          do emit switch_channel = (filter_2000,demod_qpsk)
          then UMTS
      | UMTS ->
          do until (gsm y) then Reconfigure_GSM
      end
```

Questions ?...