An Effect Type System for Modular Distribution of Dataflow Programs

Gwenaël Delaval¹  Alain Girault¹  Marc Pouzet²

¹INRIA Rhône-Alpes, Pop-Art project
²LRI, Demons team

June 13, 2008 — LCTES, Tucson, Arizona
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
  \[\Rightarrow\] separate design of closely related components, risks of data inconsistency.

- One computing resource can be involved in several functions
  \[\Rightarrow\] 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 $\Rightarrow$ separate design of closely related components, risks of data inconsistency.
- One computing resource can be involved in several functions $\Rightarrow$ 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
A reactive system is a system which reacts to its environment at the speed of the environment.
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**

![Diagram of Synchronous Reactive Systems](image_url)
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**

  ![Diagram](image)

- **Instantaneous broadcast of events**
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

![Diagram showing discrete time scale and instantaneous broadcast of events]

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

Example

```plaintext
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 Synchroné
- $x$, $\text{reset}$ and $s$ are data streams
- state of the system represented by the memories ($\text{pre}(e)$)
### Example

```plaintext
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
- \( x, \text{reset} \) and \( s \) are *data streams*
- state of the system represented by the *memories* \( \text{pre}(e) \)
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)
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:
  
  \[ y = f(x) \text{ at } A \ldots \]

- Inference of the localization of each value and computation from expressed ones: “coloration” of the program

[Caspi, Girault & Pilaud, IEEE TSE 1999]
Motivations

Semantics

Spatial types

Distribution

SDR example

Results

Conclusion

Distribution of synchronous dataflows programs

**Principles**

- Allowing the explicit localization of computations:
  \[ y = f(x) \text{ at } A \ldots \]

- 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:

  ```plaintext
  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 \( \hat{R} \)):
  - \( i \text{ at } s \) : immediate value located at \( s \)
  - \( (vl, vl') \text{ at } A \) : pair entirely located at \( A \)
  - \( (vl \text{ at } A, vl' \text{ at } B) \) : distributed pair

- **Formalization of distributed reactions involving a set of sites** \( \ell \) : \( \hat{R} \models^\ell e \xrightarrow{\hat{v}} e' \)
Distributed execution I

- An immediate value can be emitted anywhere:

\[(\text{Imm})\]

\[
\hat{R} \models \{s\} i \xrightarrow{s} i
\]

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

\[(\text{Op})\]

\[
\hat{R} \mid \hat{l_1} e_1 \xrightarrow{i_1 \text{ at } A} e'_1 \quad \hat{R} \mid \hat{l_2} e_2 \xrightarrow{i_2 \text{ at } A} e'_2
\]

\[
i = \text{op}(i_1, i_2)
\]

\[
\hat{R} \mid \hat{l_1 \cup l_2} \text{op}(e_1, e_2) \xrightarrow{i \text{ at } A} \text{op}(e'_1, e'_2)
\]
Distributed execution II

- The reaction of a \( e \) at \( A \) expression must involve at most the location \( A \):

\[
(At) \quad \hat{R} \models \{A\} e \xrightarrow{\text{\hat{v}}} e' \quad \hat{R} \models \text{at } A \xrightarrow{\text{\hat{v}}} e' \text{ at } A
\]

- A value located on one site \( A \) can be communicated on any location available from \( A \):

\[
(Comm) \quad \hat{R} \models^\ell \ x\xrightarrow{\text{vl at } A} e' \quad (A, A') \in \mathcal{L} \quad \quad \quad \hat{R} \models^\ell\cup\{A'\} \ x\xrightarrow{\text{vl at } A'} e'
\]
Type and effect system [Talpin & Jouvelot, JFP 1992]:

- $t\text{ at } s$ is the spatial type of a value available at location $s$ (either a constant location $A$ or a location variable $\delta$)

- Functional values: spatial types of the form $s_i \xleftarrow{\ell, T} s_o$ s.t. :
  - $s_i$ is the spatial type of the function’s inputs
  - $s_o$ is the spatial type of the function’s outputs
  - $\ell$ 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 $f(x) = y_3$ where

$y_1 = f_1(x)$ at $P$

and $y_2 = f_2(y_1)$ at $Q$

and $y_3 = f_3(y_2)$ at $R$

Assuming that $f_1$, $f_2$, and $f_3$ are computable everywhere:

- $f_1$ gets the spatial type $c$ at $P$ $\rightarrow \langle \{P\}, \emptyset \rangle \rightarrow c$ at $P$
- $f_2$ gets the spatial type $c$ at $Q$ $\rightarrow \langle \{Q\}, \emptyset \rangle \rightarrow c$ at $Q$
- $f_3$ gets the spatial type $c$ at $R$ $\rightarrow \langle \{R\}, \emptyset \rangle \rightarrow c$ at $R$

because of the presence of the two communications, $f$ is finally of spatial type $c$ at $P$ $\rightarrow \langle \{P, Q, R\}, [P \xrightarrow{1} Q, Q \xrightarrow{2} R] \rangle \rightarrow c$ at $R$
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 $\ell$
- the set of named communication channels $T$
Motivations | Semantics | **Spatial types** | Distribution | SDR example | Results | Conclusion
---|---|---|---|---|---|---

**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:

$$(At) \quad H \; \text{at} \; A \vdash e : t/\{A\}/T \quad A \in S$$

$$\frac{H \vdash e \; \text{at} \; A : t/\{A\}/T}{H \vdash e \; \text{at} \; A : t/\{A\}/T}$$
Communications are inserted by means of **subtyping**:

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

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

```plaintext
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 : $b$ at $A \rightarrow \langle \{A, B\} / [A \rightleftharpoons B] \rangle \rightarrow b$ 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: \( b \) at A \( \langle \{A, B\} / [A \leftarrow c \rightarrow B] \rangle \rightarrow b \) at B

\( \rightarrow \) 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 at A \langle\{A, B\}/[A \mapsto c \mapsto B]\rangle b at B

\rightarrow On A, c is added as an output:
node f(x) = ((), c) where y = x + 1 and c = y

\rightarrow On B, c is added as an input:
node f(x, c) = z where z = c + 2
Distribution example: node instanciation

\[
\text{node } g(x_1, x_2) = (y_1, y_2) \text{ where } \\
y_1 = f(x_1) \\
\text{and } y_2 = f(x_2)
\]

\[
g : (b \text{ at } A \times b \text{ at } A) \rightarrow \langle \{A, B\} / [A \overset{c_1}{\leftrightarrow} B, A \overset{c_2}{\leftrightarrow} B] \rangle \rightarrow (b \text{ at } B \times b \text{ at } B)
\]
**Distribution example: node instanciation**

\[
\text{node } g (x_1, x_2) = (y_1, y_2) \quad \text{where}
\]
\[
y_1 = f(x_1)
\]
\[
y_2 = f(x_2)
\]

\[
g : (b \text{ at } A \times b \text{ at } A) \rightarrow \{A, B\}/[A \overset{c_1}{\leftrightarrow} B, A \overset{c_2}{\leftrightarrow} B] \rightarrow (b \text{ at } B \times b \text{ at } B)
\]

\[\Rightarrow \text{use of two distinct channels: one for each instanciation}\]
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 $g((x_1,x_2),c_1,c_2) = ((x_1,x_2),c_1,c_2)$ where
  $$(x_1,c_1) = f(x_1)$$
  and $$(x_2,c_2) = f(x_2)$$

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 $g((x_1,x_2),c_1,c_2) = (y_1,y_2)$ where
  $$y_1 = f(x_1,c_1)$$
  and $y_2 = f(x_2,c_2)$
node channel(filter,demod,x) = y where
  m = filter(x) at FPGA
and y = demod(m) at DSP

channel : ∀α, β, γ, η.(α at FPGA ←⟨{FPGA}/∅⟩→ β at FPGA)
  × (β at DSP ←⟨{DSP}/∅⟩→ γ at DSP)
  × α at FPGA
  ←⟨{FPGA,DSP}/[FPGA ↪ DSP]⟩→ γ at DSP
Control: SDR example

\[
\text{node multichannel_sdr}(x) = y \quad \text{where} \\
\quad c = g(y) \\
\quad \text{and match (true fby c) with} \\
\quad | \text{true} \rightarrow y = \text{channel}(\text{filter_bp_1800, demod_GMSK, x}) \\
\quad | \text{false} \rightarrow y = \text{channel}(\text{filter_bp_2000, demod_QPSK, x})
\]

\[
\text{multichannel_sdr} : c \text{ at FPGA} \xrightarrow{\langle\{\text{FPGA, DSP}\}/T\rangle} c \text{ at DSP}
\]

with \( T = [\text{FPGA} \xleftarrow{c_1} \text{DSP}, \text{FPGA} \xrightarrow{c_2} \text{DSP}, \text{DSP} \xrightarrow{c_3} \text{FPGA}] \)
Results

Theorems

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

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

$$H \vdash D : H'/\ell/T \xrightarrow{A_1} D_1$$

$$\vdots$$

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

Then $D$ is semantically equivalent with $D_1$ and $\ldots$ and $D_n$. 
Theorems

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

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

Then $D$ is semantically equivalent with $D_1$ and $\ldots$ 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 !!!

```plaintext
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
```
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 ?...