Generating Distributed Programs from Event-B Models

Horatiu Cirstea, Alexis Grall, Dominique Méry
LORIA UMR 7503, Université de Lorraine
France

International Workshop on
Verification and Program Transformation
March 27-28,2021

Distributed Algorithms: specification and execution

Specify, design, verify and execute distributed algorithms

- `Event-B`:
  - used to model different systems at different levels of abstraction
  - refinement as a relation over `Event-B` models
  - correct by construction process for distributed algorithms
  - associated proof tools

- `DistAlgo`
  - executable
  - precise formal semantics

- **Goal**
  - Generate correct by construction code from localized `Event-B` models

- **Propose**
  - a methodology for specifying distributed algorithms in `Event-B`
  - **a restricted language `LB` (Local `Event-B`)**
  - **a tool to translate into `DistAlgo`**

Summary

MACHINE
Mch
SEES
Ctx
VARIABLES
x
INVARIANT
$I(x)$
INITIALISATION
$Init(x)$
EVENTS
$\vdots$
e
$\vdots$
END

Ctx defines the static environment for the proofs related to **Mch**: sets, constants, axioms, theorems.

$$\forall x \in Values : Init(x) \Rightarrow I(x)$$

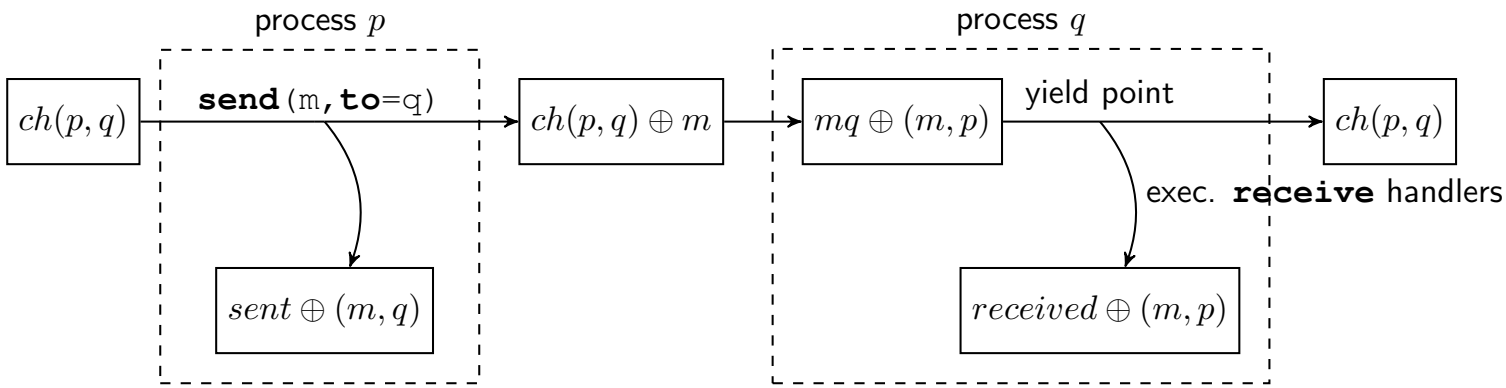
e
any
u
where
$G(x, u)$
then
$x :| (R(u, x, x'))$
end

e is **observed**: $x \xrightarrow{e} x'$

For any event e,
$\forall x, x', u \in Values : I(x) \wedge G(x, u) \wedge R(u, x, x') \Rightarrow I(x')$

Extension of Python providing high level distributed programming mechanisms

- sets of processes
- communication primitives



- boolean functions **each** and **some** acting as universal and existential quantifiers

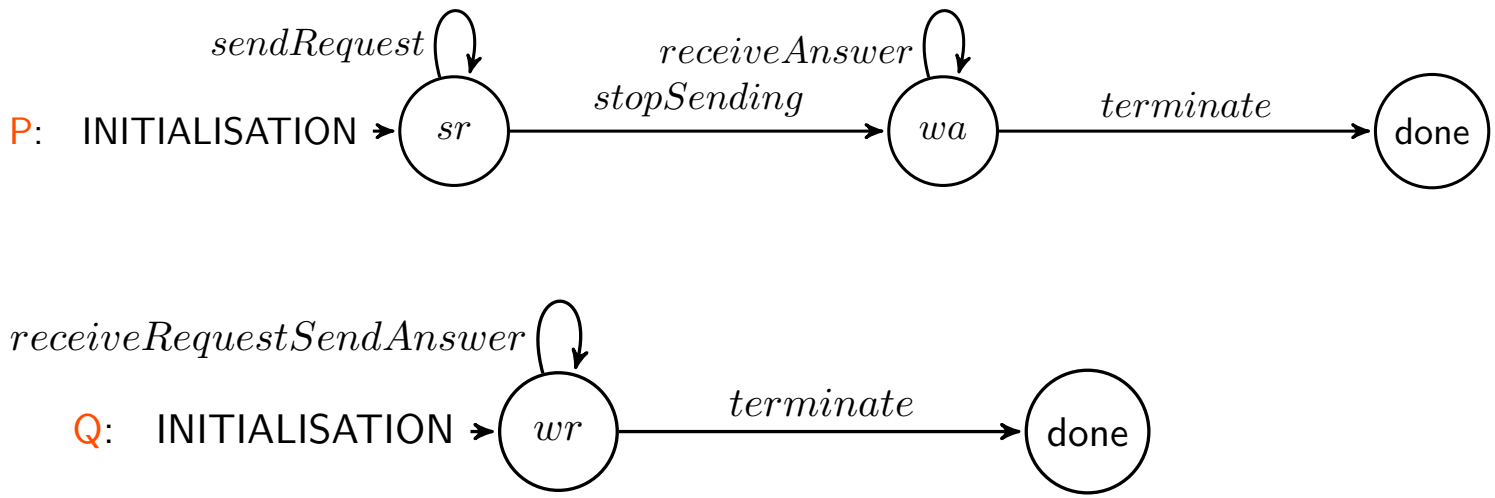
Example of DistAlgo

```
def main():
    PSet = new(P,num=cfg.NP)
    QSet = new(Q, num=cfg.NQ)
    network = {proc:QSet for proc in PSet}
    for proc in PSet:
        setup({proc}, (network[proc],))
    start(Nodes)

class P(process):
    def setup(neighbours):
        self.pc = "start"
        self.result = {}
    def run():
        while(pc != "done"):
            stateFunctions[self.pc]()
    def sending_requests():
        send("req4msg", to=q)
        self.pc = "waiting_answers"
```

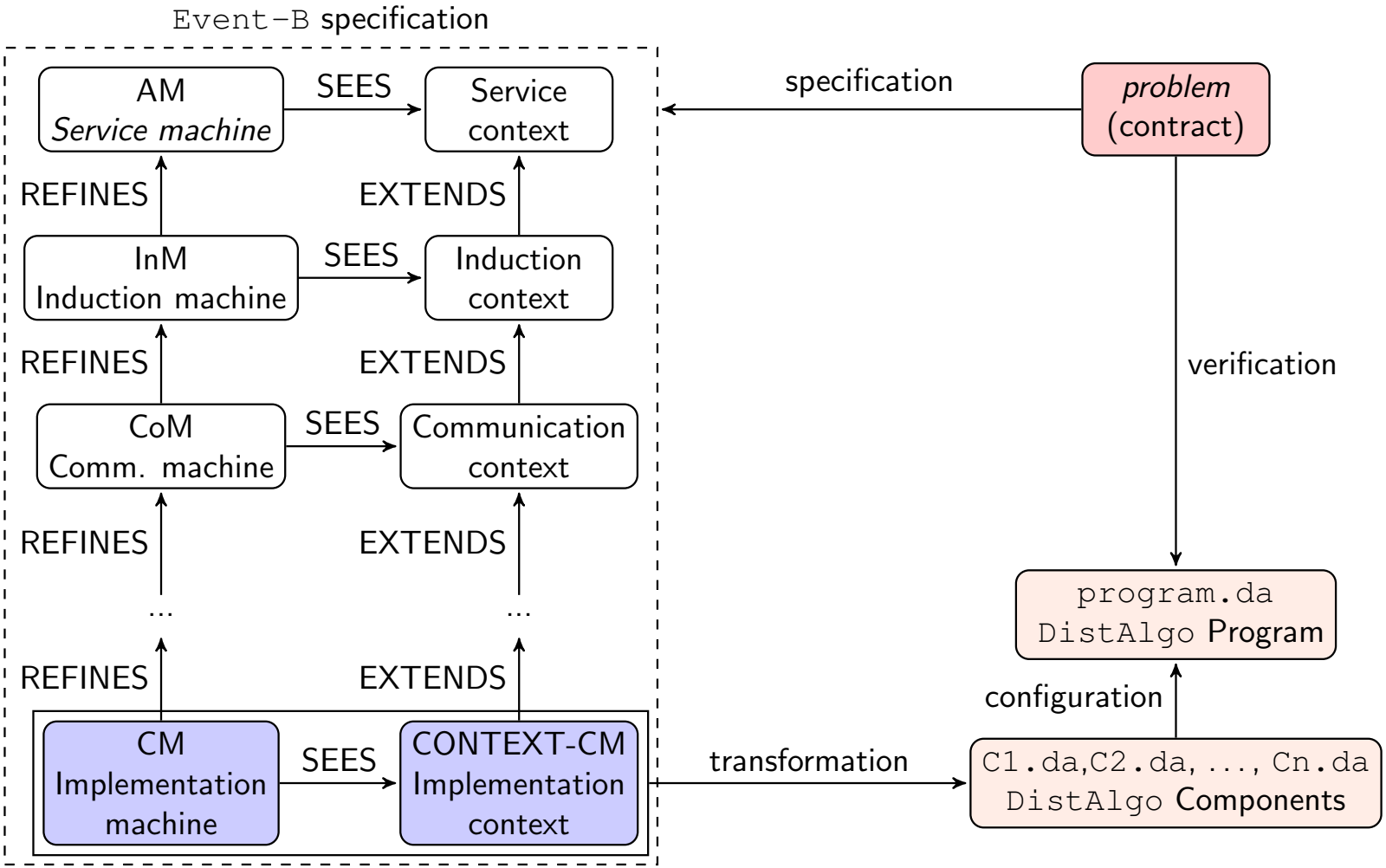
Modelling approach

- use Tel's modelling technique: a distributed algorithm is as a set of local algorithms, each local algorithm being able to either
  - do an internal action, or
  - send a message, or
  - receive a message



- use a subset of the modelling language `Event-B : LB`

Global methodology for correct-by-construction distributed algorithms



General architecture of a distributed algorithms

- the set Nodes of processes partitioned into process classes

$$\texttt{Nodes} : partition(\text{Nodes}, PCl_1, \ldots, PCl_n)$$

- the processes in each process class PCl_i

$$\texttt{PCl}_i : partition(PCl_i, \{proc_1\}, \ldots, \{proc_m\})$$

- the topology specified by a function network $\in$ Nodes $\to \mathbb{P}$ (Nodes) associating to each process its neighbours:

$$\texttt{network\_val} : \text{network} = \{proc \cdot proc \in PCl_1 | proc \mapsto expr_1\} \cup \\ \cdots \cup \{proc \cdot proc \in PCl_n | proc \mapsto expr_n\}$$

Communication channels

- Channels: communication channels between processes
- Messages: the set of messages exchanged through channels
- state of a channel: multiset of messages sent, received, in transition

$$\text{sent}, \text{received}, \text{inChannel} : \text{Channels} \times (\text{Nodes} \times \text{Nodes}) \times \text{Messages} \to \mathbb{N}$$

$$\text{send}, \text{receive}, \text{lose} : \text{Channels} \times (\text{Nodes} \times \text{Nodes}) \times \text{Messages} \to \text{Channels}$$

General context

CONTEXT CONTEXT-CM
EXTENDS C00
SETS
Nodes Messages
...
CONSTANTS
network
Channels emptyChannel
sent received inChannel
send receive lose
P p Q
...
AXIOMS
$partition(\text{Nodes}, P, Q)$
$partition(P, \{p\})$
$\text{network} \in \text{Nodes} \to \mathbb{P}(\text{Nodes})$
$\text{network} = \{proc \cdot proc \in P | proc \mapsto Q\} \cup \{proc \cdot proc \in Q | proc \mapsto \{p\}\}$
$\text{Channels} = \text{Nodes} \times \text{Nodes} \to (\text{Messages} \to \mathbb{N} \times \mathbb{N} \times \mathbb{N})$
...
END

Algorithm specific resources

A constant cst is *local* to a process $proc \in PCl$ when it is

- a function whose evaluation depends on $proc$
  (i.e. of type $PCl \rightarrow cstType$ or Nodes $\rightarrow cstType$)

  CONSTANTS

  $availableResources$

  AXIOMS

  $availableResources \in Q \rightarrow \mathbb{N}$

  network $\in$ Nodes $\rightarrow \mathbb{P}$(Nodes)

- an element of an enumerated set annotated by some $@PCl$

  SETS

  $MessagePrefixes$

  CONSTANTS

  $request\ answer$

  AXIOMS

  $partition(MessagePrefixes, \{request\}, \{answer\})$ //@P @Q

Local Algorithms

The machine CM declares the types and initializes the *local variables* (of type $PCl \to varType$ or Nodes $\to varType$) of each process class.

MACHINE CM
SEES CONTEXT-CM
VARIABLES
$chans \quad pc \quad result$
INVARIANTS
$chans \in$ Channels
$pc \in$ Nodes $\to STATES$
$result \in P \to ($Nodes $\nrightarrow \mathbb{N})$
EVENTS
Initialisation $\widehat{=}$
begin
$chans :=$ emptyChannel
$pc := \{proc \cdot proc \in P | proc \mapsto sr\} \cup \{proc \cdot proc \in Q | proc \mapsto wr\}$
$result := \{proc \cdot proc \in P | proc \mapsto \varnothing\}$
end
END

Describe the algorithm: Events

Each *event* evt of the machine CM corresponds to state transitions of the local algorithms of the processes and it features

- one parameter $proc \in PCl$ with $PCl \in$ Nodes
- a guard $pc(proc) = st$ (specifies the event is enabled in $st \in$ States)
- a typing guard $t \in tExpr$ for each parameter

Event $\mathbf{sendRequest} \mathrel{\widehat{=}}$
  any $\mathbf{p}$ q
  where
    $\mathbf{p} \in \mathbf{P}$
    $pc(\mathbf{p}) = \mathbf{sr}$
    $q \in$ network(p)
  then
    ...
  end

$\mathbf{sendRequest}$ *observed* for $\mathbf{P}$, *observable* in state $\mathbf{sr}$.

Describe the algorithm: Events

Each *event* evt is such that

- it features
  - general guards
  - sent$(chans \mapsto (proc \mapsto pExpr) \mapsto mExpr) = nExpr$
  - received$(chans \mapsto (pExpr \mapsto proc) \mapsto mExpr) = nExpr$
- all actions are assignments
  - $x(proc) := pExpr$
  - $chans :=$ send$(chans \mapsto (proc \mapsto pExpr) \mapsto mExpr)$
  - $chans :=$ receive$(chans \mapsto (pExpr \mapsto proc) \mapsto mExpr)$

Event $sendRequest \mathrel{\widehat{=}}$
  any $p\ q$
  where
    $\ldots$
    sent$(chans \mapsto (p \mapsto q) \mapsto request) = 0$
  then
    $chans :=$ send$(chans \mapsto (p \mapsto q) \mapsto request)$
  end

Translation

The `LB` specification consisting of

- a machine CM
- a context CONTEXT-CM

is translated towards a `DistAlgo` program composed of a set of process:

- axioms in CONTEXT-CM $\Longrightarrow$ main function and process classes
- invariants and events in CM $\Longrightarrow$ process class methods

Expressions

- arithmetic expressions
  `Python` operations

- set expressions
  `Python` primitives (**`set`**, **`setof`**) and operations

- send message, $chans := \mathsf{send}(chans \mapsto (proc \mapsto dest) \mapsto msg)$
  `send`$(\overline{msg}$**`, to=`**$\overline{dest})$

- queries, $\mathsf{received}(chans \mapsto (source \mapsto proc) \mapsto msg) > 0$
  `some`(**`sent`**$(\overline{msg}$**`,to=`**$\overline{dest})$)

Main function

Process classes

Nodes : $partition(\text{Nodes}, PCl_1, \ldots, PCl_n)$ $\Longrightarrow$

```
PClSet_1 = new(PCl_1, num=NPCl_1)
...
PClSet_n = new(PCl_n, num=NPCl_n)
Nodes = set.union(PClSet_1,...,PClSet_n)
```

Topology

network_val : network $= \{proc \cdot proc \in PCl_1 | proc \mapsto expr_1\} \cup$
$\cdots \cup \{proc \cdot proc \in PCl_n | proc \mapsto expr_n\}$ $\Longrightarrow$

```
network = {proc:expr_1 for proc in PClSet_1}
network.update({proc:expr_2 for proc in PClSet_2})
...
network.update({proc:expr_n for proc in PClSet_n})
```

Main DistAlgo

AXIOMS

Nodes: $partition(\text{Nodes}, P, Q)$

network_val:

network $= \{proc \cdot proc \in P | proc \mapsto Q\} \cup \{proc \cdot proc \in Q | proc \mapsto \{p\}\}$

```
def main():
    PSet = new(P,num=cfg.NP)
    QSet = new(Q,num=cfg.NQ)
    NODES = set.union(PSet, QSet)

    network = {proc:QSet for proc in PSet}
    network.update({proc:{p} for proc in QSet})
```

Initialisation

Local constants initialisation ($cst \in PCl \to cstType$)

`cst_value` : $cst = \{proc \cdot proc \in PCl | proc \mapsto expr\}$ $\Longrightarrow$

```
cst = {proc:expr for proc in PClSet}
```

Process classes initialisation

`Nodes` : $partition(\text{Nodes}, PCl_1, \ldots, PCl_n)$ $\Longrightarrow$

```
for proc in PClSet_i:
    setup({proc}, (cst_1[proc],...,cst_n[proc])
```

---

with $cst_1, \ldots, cst_n$ local constants of PCl_i

Process execution

```
start(Nodes)
```

AXIOMS

Nodes: $partition(\text{Nodes}, P, Q)$

```
def main():
    NODES = set.union(PSet, QSet)
    ...
    for proc in PSet :
        setup({proc}, (network[proc]))
    for proc in QSet :
        setup({proc}, (network[proc],
              availableResources[proc]))

    start(NODES)
```

Setup method

Nodes : $partition(\text{Nodes}, PCl_1, \ldots, PCl_n) \implies$

```
class PCl( process ) :
    def setup(cst_1, ..., cst_n):
        self.var_1 = expr_1
        ⋮
        self.var_m = expr_m
```

---

with $var_i := \{proc \cdot proc \in PCl | proc \mapsto expr_i\}$ (local variables of PCl)

Control flow

```
def run():
    stateFunctions = {"st1":st1,...,"stn":stn}
    while(self.pc!=done):
        stateFunctions[self.pc]()
```

---

with sti the states of PCl

Process classes

```
class P(process):
    def setup(network):
        self.pc = "sr"
        self.result = {}

    def run():
        stateFunctions = {
          "wa":wa,
          "sr":sr }
        while(pc != "done"):
            stateFunctions[self.pc]()
```

Guards and actions

Event $evt \mathrel{\widehat{=}}$
  any $\mathbf{p} \ldots$
  where
    $\mathbf{p} \in \mathbf{P}$
    $pc(\mathbf{p}) = \mathbf{st}$
    $t_1 \in S_1, \ldots, t_l \in S_l$
    $g_1, \ldots, g_n$
  then
    $a_1, \ldots, a_m$
  end

$\overline{\text{Guards}(evt)}$

`self.pc=="st"` **and**
some(t_1 **in** $\overline{S_1}$, ..., t_l **in** $\overline{S_l}$,
  has=$\overline{g_1}$ **and** ... **and** $\overline{g_n}$)

$\overline{\text{Actions}(evt)}$

$\overline{a_1}$
...
$\overline{a_m}$

Methods

Use events $\{evt_1, \ldots, evt_m\}$ *observable* in state st for PCl to generate

---

```
--st
def st():
    if await (Guards(evt_1)):
        Actions(evt_1)
    ...
    elif(Guards(evt_m)):
        Actions(evt_m)
    elif(self.pc != "st"):
        pass
```

Methods

```
class P(process):
  def run():
      stateFunctions = {"wa":wa, "sr":sr }
      while(pc != "done"):
          stateFunctions[self.pc]()

  def sr():
      # event sendRequest
      if(self.pc == "sr" and
        some(q in self.network,
           has=not(some(sent((request,), to=_q))))):
        send((request,), to=q)
      # event stopSending
      elif(self.pc == "sr" and
         each(q in self.network,
            has=some(sent((request,), to=_q)))):
        self.pc = "wa"
      elif(self.pc != "sr"):
        pass
```

Conclusions and future work

- What has been done
  - Definition of a sublanguage `LB` for localizing `Event-B`.
  - Relationship between the *modelling* language `Event-B` and the *distributed programming* language `DistAlgo`
  - Compiler towards `DistAlgo` integrated into the `Rodin` platform

- What remains to be done
  - Extend the transformation of `LB` models towards other programming languages (MPI, Go, ...)
  - Improve the transformation plugin
  - More flexible concrete modelling language