Scribbling Interactions with a Formal Foundation

Kohei Honda (QM), Aybek Mukhamedov (QM),
Gary Brown (Red Hat), Tzu-Chun Chen (QM)
and Nobuko Yoshida (IC)

February 9, 2011

ICDCIT'2011, Bhubaneswar

In collaboration with:

Matthew Arrott (OOI)

David Frankel (SAP)

Matthew Rawlings (ISO TC68 WG4/5)

Alexis Richardson (VMware)

Steve Ross-Talbot (Cognizant)

and all our academic colleagues.

Outline

- Large-Scale Distributed Applications and π-calculus
- Scribble: key ideas
- Examples
- The π-calculus in action
- Use Case: Ocean Observatory Initiative
- Current Status and Perspective

Building and Understanding

- Computing science is about both understanding and building.
- Understanding computation through theories.
- Using that understanding for building.

π-Calculus [MPW89, M90]

Grammar of an asynchronous fragment [B91, HT91]:

$$P \;::=\; x(y).P \mid \overline{x}\langle y\rangle \mid P|Q \mid (\nu x)P \mid \;\; \mid\; !x(y).P \mid \mathbf{0}$$

Reductions are asynchronous name passing:

$$\begin{aligned} x(y).P\,|\,\overline{x}\langle z\rangle \quad &\longrightarrow \quad P[z/y] \\ !x(y).P\,|\,\overline{x}\langle z\rangle \quad &\longrightarrow \quad !x(y).P\,|\,P[z/y] \end{aligned}$$

Processes only pass links, and that's enough.

What is the π-Calculus?

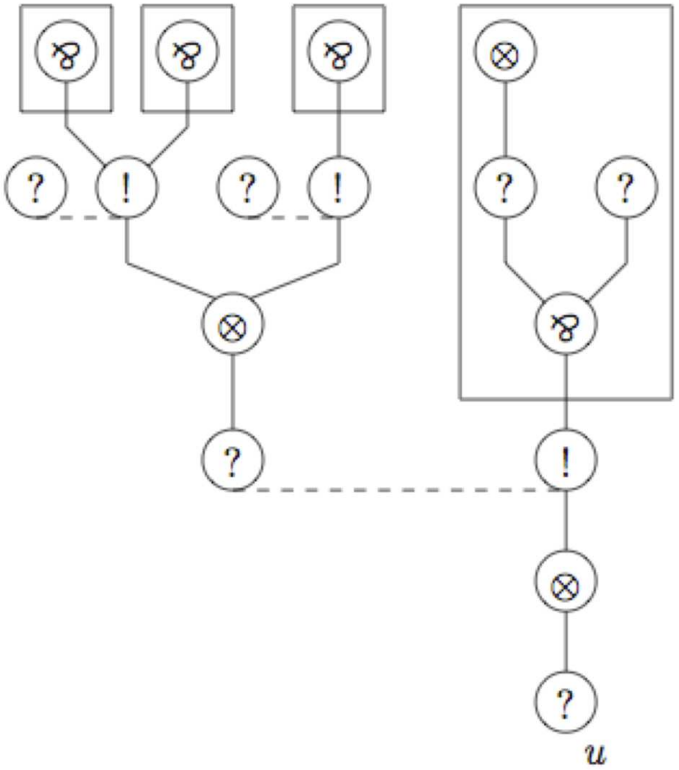
- A basic calculus of interaction (e.g. combinators) with rich theories, in the tradition of ACP, CSP and CCS (cf. Petri Nets).
- A tool for representing and analysing computational behaviour: “a descriptive tool for computing science” (Milner, 1998)
- A core communication-based programming language for experimenting new ideas.

Pi-Calculus and Polarised Proof Nets

Proofs are processes and vice versa [A94, HL10, EL07].

$$(\nu x)(\overline{u}\langle x\rangle \mid \overline{x}(yz)(!y.0 \mid !y.0 \mid !z.0) \mid !x(yz).\overline{y})$$

becomes:



Session Types (1994/1999-)

- Application-level distributed interactions are often organised as multiparty sessions (“conversations”).
- Session types describe these conversations as types.
- One way to “divide and conquer” distributed computing.
- Associated language primitives, statically checkable.
- Ensures communication safety (cf. synchronisation bug).

Some Work on Session Types (1)

- First-order sessions [THK94]
- Higher-order session [HVK98]
- Subtyping (Gay, Padovani, ..) [GH00, GH05, P09, ..]
- Integration with objets [DMYD06,GVRGC10]
- Multiparty session types [BC07,HYC07,BCCDDY08,CDGP09,...]
- Parametrised/dynamic session types [YDBH10,DY11,..]

Some Work on Session Types (2)

- Integration with Logic [ICALP'08,TGC09,CONCUR10]
- Security concerns (Malo,...) [CSF07, CONCUR10]
- Exception [CONCUR09,CONCUR10]
- Haskell (Neubauer, Thiemann) [PADL04]
- Microsoft Singularity Operating System (Fähndrich et. al)[EuroSys06]
- SJ (Hu, Pernet, Yoshida, Honda) [ECOOP08/10]
- BICA (Gay, Vasco, ...) [POPL10]

WS-CDL Collaboration

Initiated by Robin Milner in 2004 for dialogue between practice (W3C WS-CDL WG) and theory (π-calculus, with Yoshida, Carbone and Honda).

- The use of theories for semantic analysis of description of “choreographies”.
- Many basic, if informal, ideas.
- Mutual enrichment.

Communication-Centred Software (1)

A few examples:

- global financial protocols
- big services such as search engines.
- cloud computing
- communication-centred OS
- manycore chips
- large-scale cyberinfrastructure (e.g. E-healthcare)

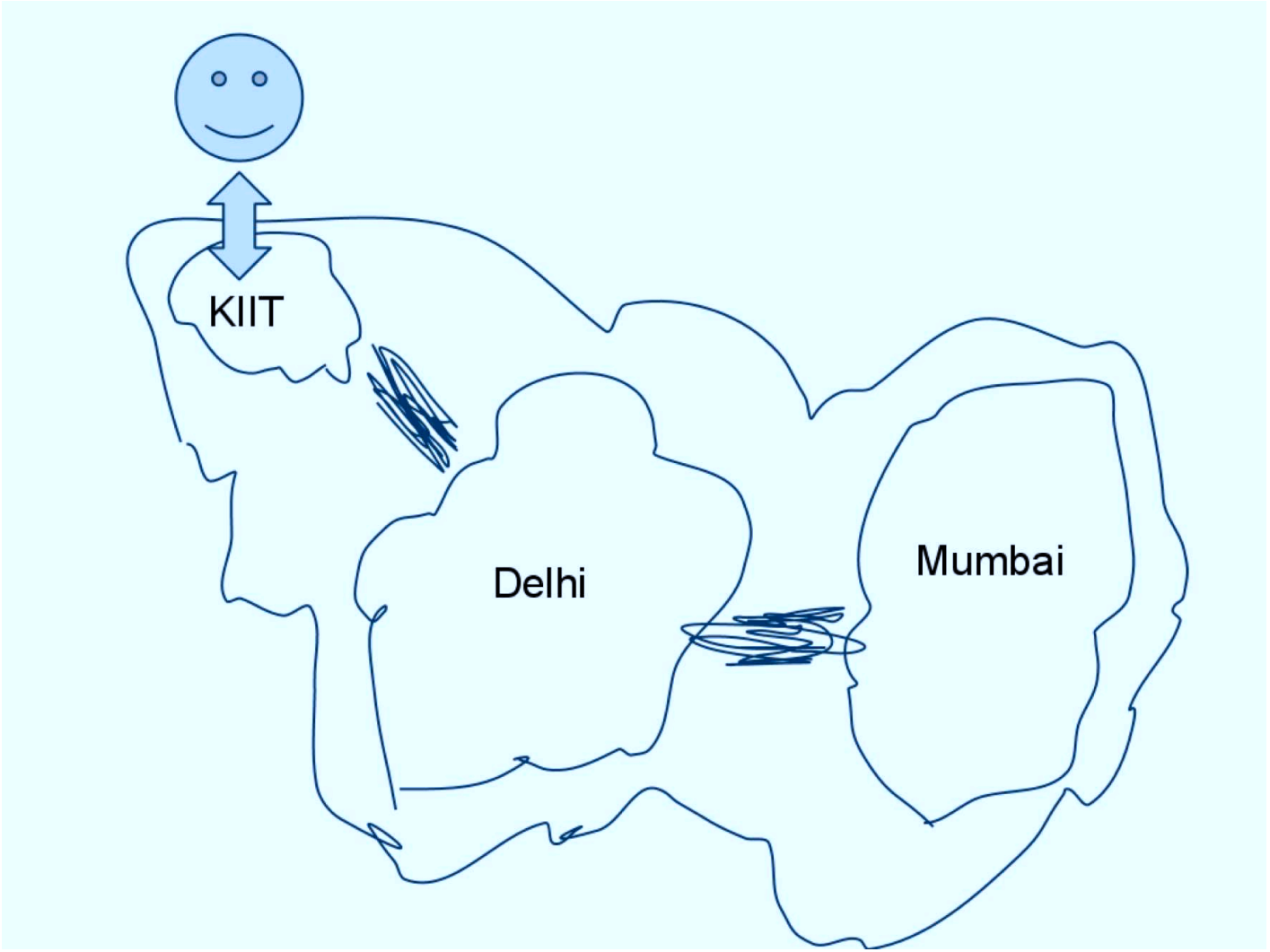
Communication-Centred Software (2)

More generally:

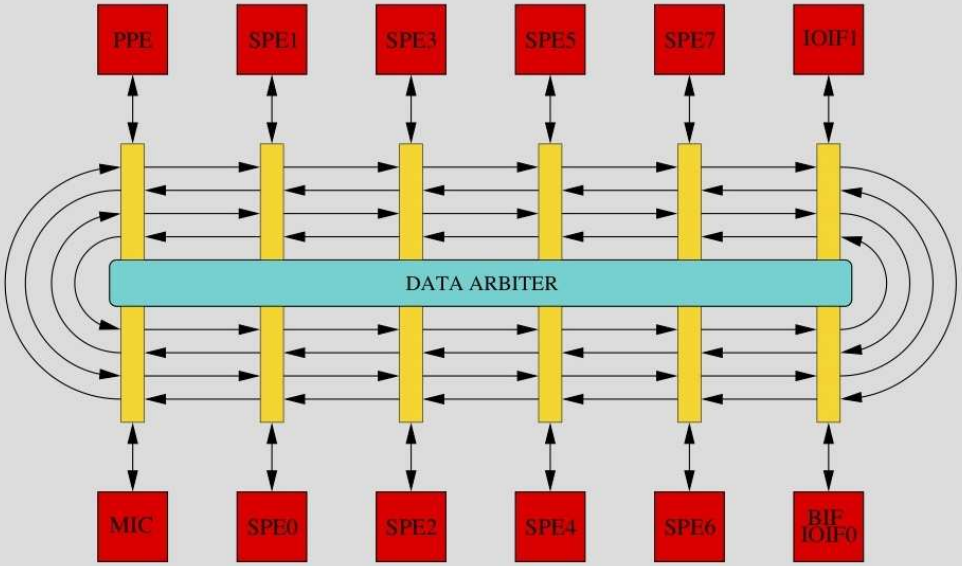
- Communication is an economical way to share data (you can pull and push without waste).
- Read/Write is communication (300 cycles).
- Communication scales (from intra-chip to clouds to Ocean).
- Communication is expressive and explicit.
- Communication is flexible and manipulable.

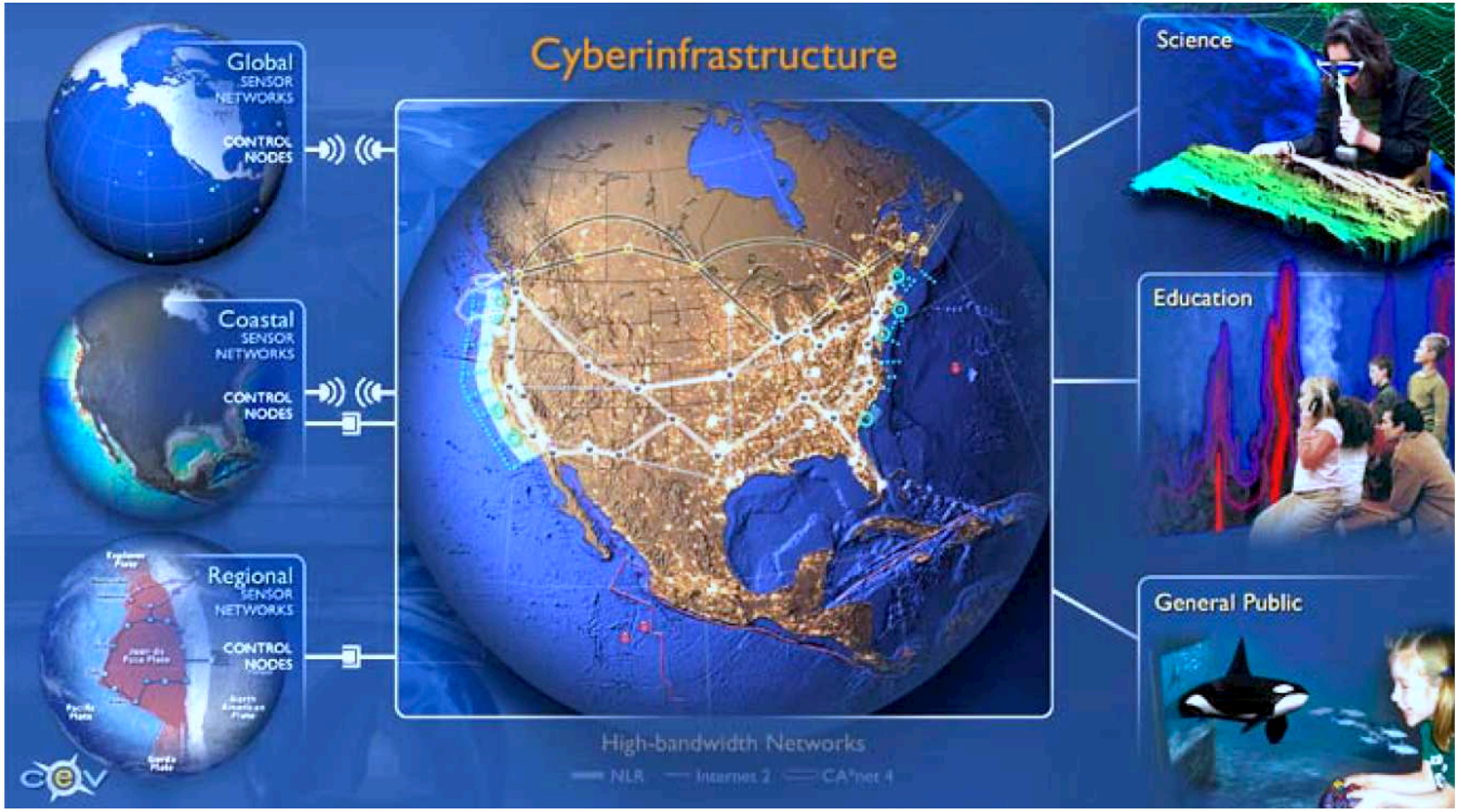
The ISO 20022 recipe is used! (1/10) Payments

Clouds



Internal Structure of the Cell BE




Cyberinfrastructure
Global
SENSOR
NETWORKS
CONTROL
NODES
Coastal
SENSOR
NETWORKS
CONTROL
NODES
Regional
SENSOR
NETWORKS
CONTROL
NODES
High-bandwidth Networks
NLR
Internet 2
CA*net 4
Science
Education
General Public

Problem: Development!

How can we develop communication-centred software?

- UML, Java, .. are highly effective for software based on objects and operations on objects.
- But they do not offer a descriptive framework for interactions.
- No way to build up communication-centred software from its components (e.g. APIs).

Types and Logic (1)

What is type signature?

```
int foobar(String s, int i);
```

- Minimal interface (and hard to change).
- Compositionally assures type safety.
- Efficiently checkable.

Types and Logic (2)

Annotating types by assertions (Design-by-Contract):

```
int foobar(String s, int i) {
   pre: { s.length()<=100 and i<=10 }
   post:{ result <=1000 }
}
```

- Refined interface: can be as vague/detailed as possible.
- Compositionally assures refined safety.
- Effectively runtime checkable (if we take care).

Challenges

Can we extend this framework to communications and concurrency?

- What are types for interactions?
- How can we validate asynchronously communicating processes?
- Can we make it really usable for large-scale distributed applications?

Scribble

Scribble is a trial to answer these questions:

- A PL-neutral language for describing application-level protocols.
- The logical layer(s) on the top of the type layer.
- Active collaboration with both academic and industry colleagues (the latter including prospective users).
- Will come with an Eclipse-based tool chain for developing distributed applications.

It is intended to help you to scribble interactions with a formal basis.

Scribble: Key Ideas (1)

Developer-friendly while being faithful to theories.

- Precisely follows [Bettini et al. 2008] and its extensions (but looks very different).
- All theoretical results use the π-calculus, without losing generality.
- Validation algorithms are provably sound, and usable for the tool chain.

Scribble: Key Ideas (2)

Protocol description and type structures.

- Protocols often include branch: branching (&) and selection ($\oplus$) types.
- Protocols often use repetition: recursive types.
- Protocols often need global escape: exception.

By this correspondence, type checking is tractable (polynomial).

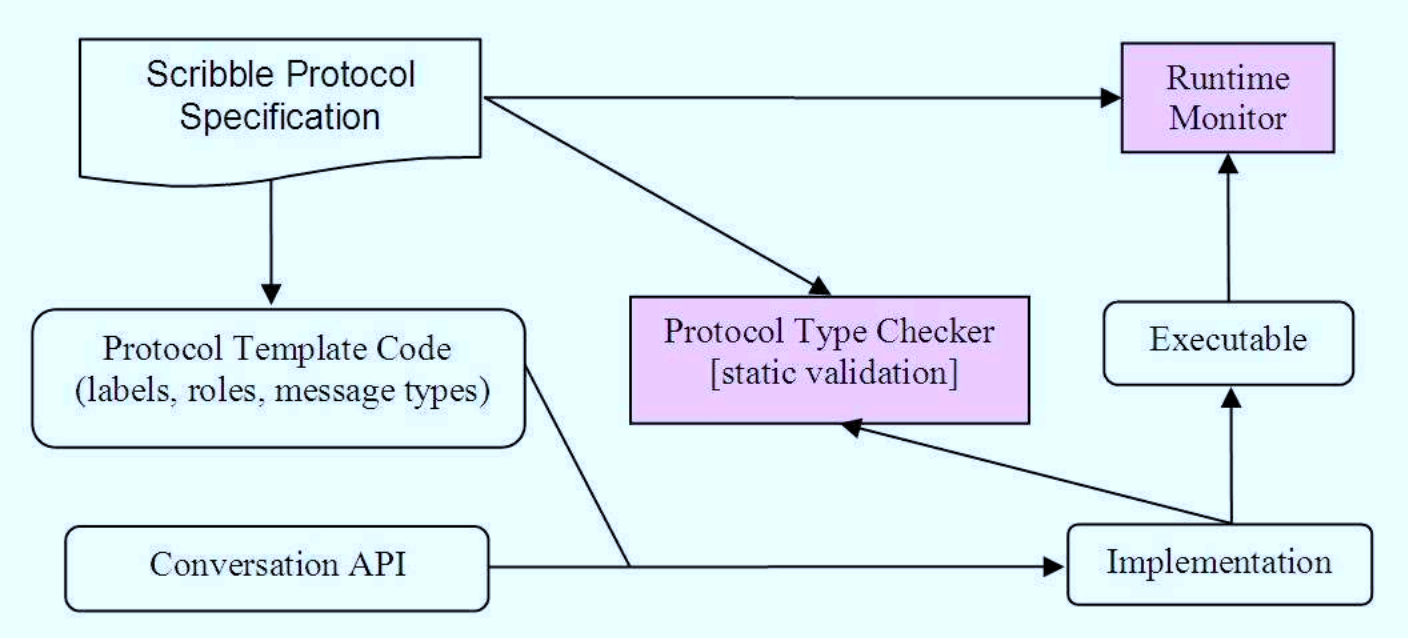
Scribble: Key Ideas (3)

How a developer uses Scribble:

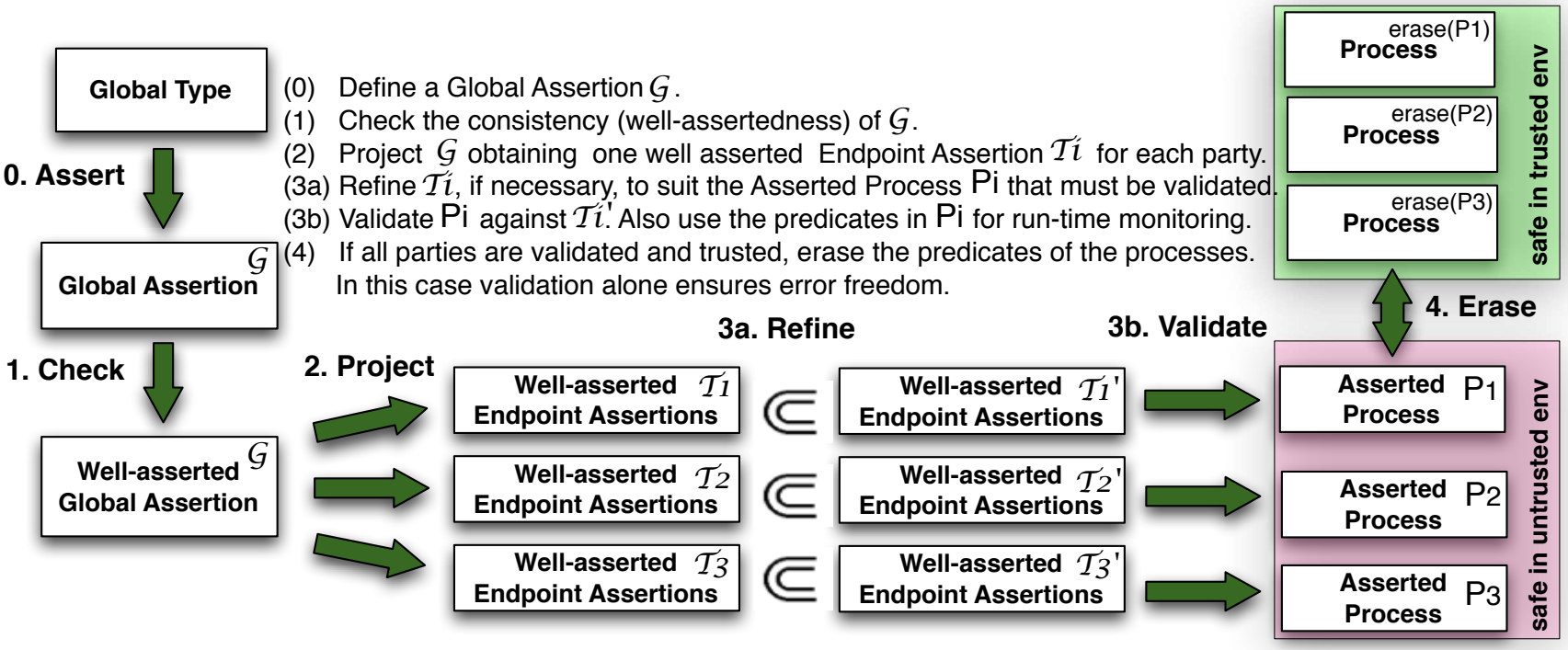
1. Author a conversational use case.
2. Specify a global protocol and validate.
3. As needed, elaborate it with assertions.
4. Project the global protocol to each endpoint and develop, with static type checking.
5. Track defects using local/global protocols.
6. Validate against specified protocols at runtime as needed.

Scribble: Key Ideas (4)

The programming for Scribble can use either API or language extensions: the following is the API-based approach (following our recent prototype).



End-point Projection



Scribble: Key Ideas (6)

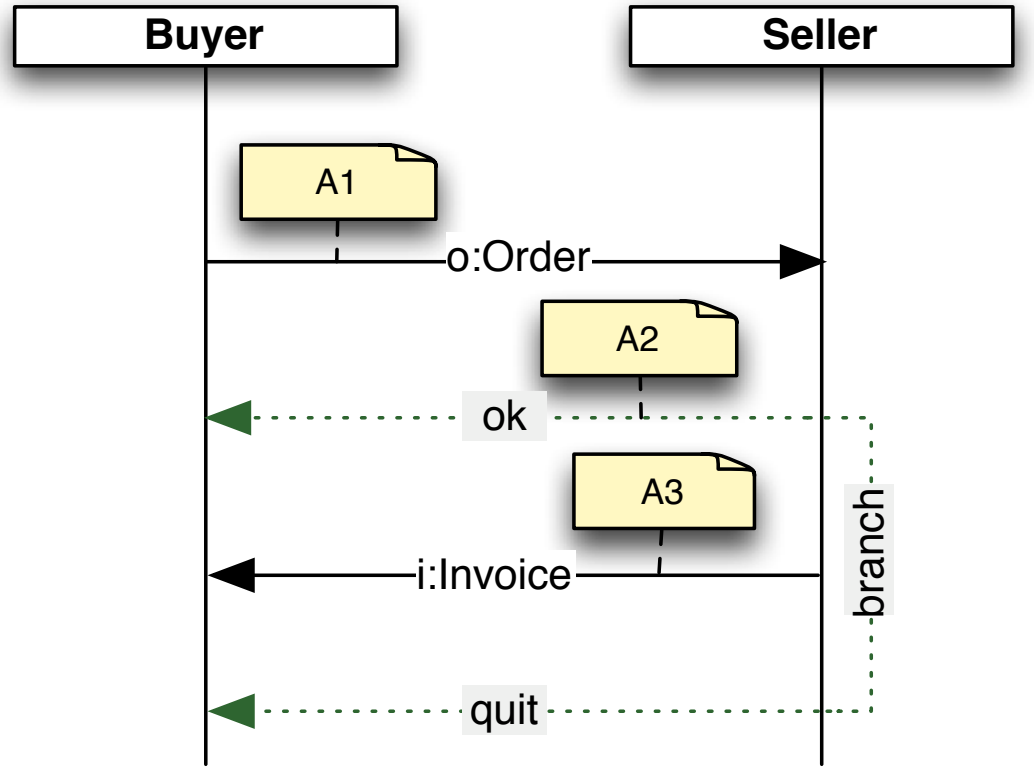
For Scribble to become usable, we need a **tool chain**, being developed as an open source project, which will:

- Assist the use of Scribble throughout the SDLC, based on Eclipse and OSGi.
- Integrate with conversation APIs/PL extensions.
- Perform parsing and semantic validations.
- Interface smoothly with existing artifacts (e.g. BPMN-2, UML, ..)

It is being developed in dialogue with prospective users.

Simple Examples (1)

```
import Order, Invoice;
protocol BuyerSeller {
    role Buyer, Seller;
    Order from Buyer to Seller;
    choice from Seller to Buyer {
        ok: Invoice from Seller to Buyer; End.
        quit: End.
    }
}
```



predicates

```
A1 = o.price < 100
A2 = o.quantity < MAXORDER
A3 = i.quantity= o.quantity ∧ i.price=o.price
```

Simple Examples (2)

```
protocol BuyerSellerBank {
    role Buyer, Seller, Bank;
    Order from Buyer to Seller;
    choice from Seller to Buyer {
        ok:
          Invoice from Seller to Buyer;
          Invoice from Buyer to Bank;
          Payment from Bank to Seller;
          End.
        quit: End.
    }
}
```

Travel Agent Example (1)

1 The client interacts with the travel agent to request information about various services.

2 Prices and availability matching the client requests are returned to the client. The client can then perform one of the following actions:

(a) The client can refine their request for information (repeat step 2). OR

(b) The client may reserve services based on the response, OR

(c) The client may quit the interaction with the travel agent.

....

Travel Agent Example (2)

3 When a customer makes a reservation, the travel agent checks the availability of the requested services and either

(a) All services are available hence are reserved. OR

(b) If not available, the client is informed.

- Either
  - i. Given alternative options for those services. OR
  - ii. Client restarts the search back in step 1.
- Go back to step 3.

4 For every relevant reserved service the travel agent takes a payment for the reservation.

5 The client is issued a reservation number for the transaction.

```
protocol ReserveTravel {
  role Client, Agent, Provider[1..num_providers];
  query(Services) from Client to Agent;
  Services_info from Agent to Client;
  rec X {
    choice from Client to Agent {
      more_info():
        query(Services) from Client to Agent;
        Services_info from Agent to Client;
        #X;
      reserve(): ...
      quit(): end;
    }
} }
```

```
reserve():
  query() from Agent to Provider[1..num_providers];
  Info from Provider[1..num_providers] to Agent;
  choice from Agent to Client {
    all_available():
      reserve(Services)
        from Agent to Provider[1..num_providers];
      run PurchaseTravel(num_providers);
    altern_services():
      Altern_services_info from Agent to Client;
      #X;
    restart():   end;
  }
```

```
protocol PurchaseTravel(num_providers) {
  rec X {
    choice from Client to Agent {
      cancel(): ...
      add_services(): ...
      book():
        Payment from Client to Agent;
        book(Services)
          from Agent to Provider[1..num_providers];
        confirm(Services)
          from Provider[1..num_providers] to Agent;
      quit(): end;
} } }
```

Formal Foundations for Type Layer

Using the π-calculus, we can formally obtain:

- An efficient algorithm for consistency validation.
- A compositional typing system.
- A sound, complete and efficient (and scalable) typing algorithm to check whether programs are type-correct or not.

Adding Assertions

A basic extension is presented in [BHTY2010].

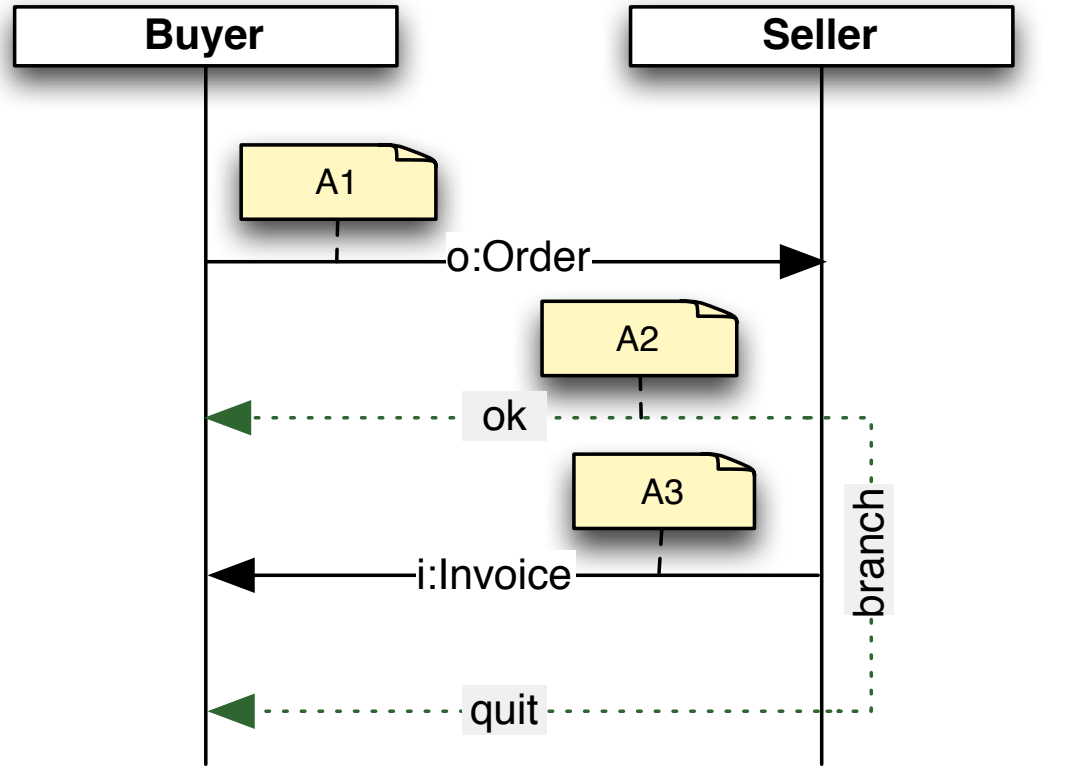
- Annotate interactions with assertions: “a sender guarantees and a receiver relies”.
- Extend end-point projection to logical assertions: “responsibility to satisfy assertions at endpoints”.
- Can treat many use cases, and serve as a basis for further extensions.

From Types...

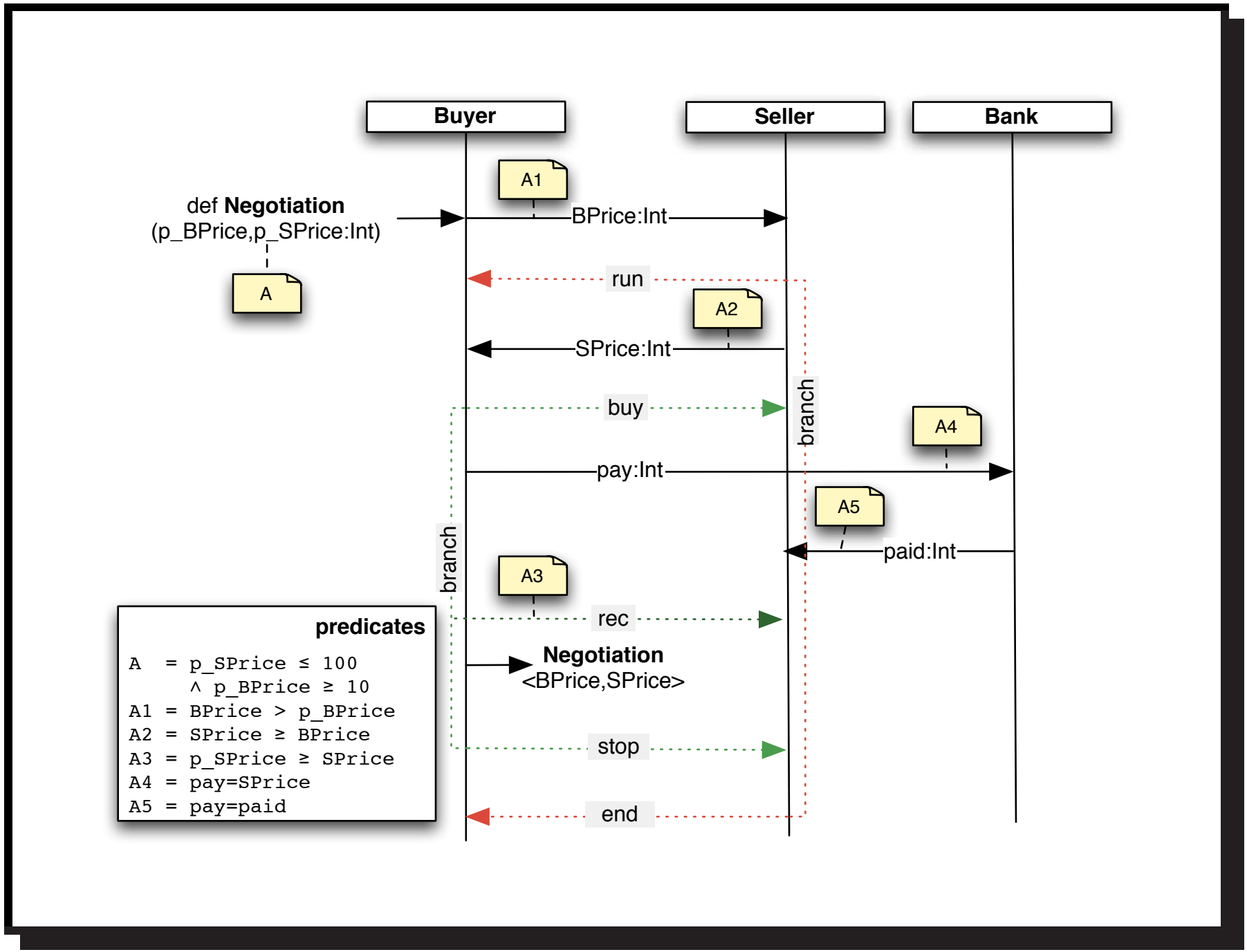
```
import Order, Invoice;
protocol BuyerSeller {
    participants Buyer, Seller;
    Order from Buyer to Seller;
    choice from Seller to Buyer {
        ok: Invoice from Seller to Buyer; End.
        quit: End.
    }
}
```

...To Assertions

```
import Order, Invoice;
protocol BuyerSeller {
    participants Buyer, Seller;
    o:Order {o.price <= 100} from Buyer to Seller;
    choice from Seller to Buyer {
        ok {o.quantity<=MaxQuantity}:
            i:Invoice {i.price == o.price &&
                       i.quantity == o.quantity}
                from Seller to Buyer; End.
        quit: End.
    }
}
```




```
A1 = o.price < 100                                    predicates
A2 = o.quantity < MAXORDER
A3 = i.quantity= o.quantity ∧ i.price=o.price
```


Buyer
Seller
Bank
A1
def Negotiation
(p_BPrice,p_SPrice:Int)
BPrice:Int
A
run
A2
SPrice:Int
buy
branch
A4
pay:Int
A5
paid:Int
branch
A3
rec
Negotiation
<BPrice,SPrice>
stop
end
predicates
A = p_SPrice ≤ 100
∧ p_BPrice ≥ 10
A1 = BPrice > p_BPrice
A2 = SPrice ≥ BPrice
A3 = p_SPrice ≥ SPrice
A4 = pay=SPrice
A5 = pay=paid

Formal Foundations for Logic Layer

We again use the π-calculus.

- Formal semantics based on typed LTS.
- A validation algorithm for consistency validation.
- A sound and complete proof system [BHTY2010] for a large class of typable processes.
- Many results rely on the underlying type structures.
- Further extensions being studied.

The π-calculus in Action

What is the point of using the π-calculus for Scribble?

- The algorithms etc. we obtain are provably sound.
- It captures essentially the whole interactional behaviour, so the technical results (e.g. proof rules) scale to real-world languages.
- We can rely on the accumulated results from existing theories including those from functional types and logics.

(details in separate slides, if we have time)

Use Case: OOI

(separate slides)

Current Status and Perspective (1)

- Scribble and its tool chain are being developed by the present authors and other colleagues, in extensive dialogue with prospective users.
- Programming language environments are being developed (with a prototype running) for Java, Scala and OCaml.
- The first stable release for the tool chain is planned at the end of 2011.

Current Status and Perspective (2)

Summary:

- Communication-centred computing is already here.
- Large-scale distributed applications are inevitably communication-centred.
- We need a general methodology for effective development and formal assurance.

When the tool is released, I hope you will find it nice to scribble interactions with a formal foundation.