Implementing, Verifying and Debugging Distributed Systems

Elisa Gonzalez Boix

https://soft.vub.ac.be/disco/
How it all started?

• A programming model for ubiquitous computing

In an ubiquitous context, asynchronous communication suits better, why don’t you implement an actor-based language?

my own language?!?
factor(n) := createActor({
    fac (c, client) :{
        if (n > 1, {
            next: factor(n-1);
            if (n = c, next.fac(c, client),
                next.fac(c*n, client))},
            client.result(c))
    };
});

factorial := createActor({
    result(res) : { display(res);},
    compute(num) :{
        a : factor(num);
        a.fac (num, thisActor());
    };
});
factorial.compute(3);
How it all started again?

- a programming model for ubiquitous computing

what about exploring a failure handling model for AmbientTalk?

AmbientTalk’s Distributed Model = OO + Events

A superset of object-oriented programming that is explicitly geared towards programming distributed applications that run on mobile ad hoc networks

- Generate and receive application requests:
  \[ \text{obj<-msg(arg)} \]
  \[ \text{def msg(param) \{ ... \}} \]

- Follow-up on outstanding requests:
  \[ \text{when: future becomes: \{ |result| ... \}} \]

- React to services appearing and disappearing:
  \[ \text{when: type discovered: \{ |ref| ... \}} \]
  \[ \text{whenever: type discovered: \{ |ref| ... \}} \]

- React to references disconnecting, reconnecting:
  \[ \text{when: ref disconnected: \{ ... \}} \]
  \[ \text{when: ref reconnected: \{ ... \}} \]
  \[ \text{whenever: ref disconnected: \{ ... \}} \]
  \[ \text{whenever: ref reconnected: \{ ... \}} \]

Distributed Applications in AmbientTalk

• Data stored at the owner, and all operations go via asynchronous message passing

```plaintext
deftype PingPong;
def pingPong := object: {
def ping(){
    system.println("received ping");
    \`pong;
}
}
export: pingPong as: PingPong;
```

```plaintext
deftype PingPong;

whenever: PingPong discovered: {
    |farRef|
    when: farRef<-ping()@FutureMessage()
    becomes: {
        |val|
        system.println("received " + val);
    }
}
```
How to reconcile failures with distributed event-based model?

“A lease denotes the right to access a resource for a specific duration negotiated when the access is first requested.”

```
when: ref disconnected: { ... }
when: ref reconnected: { ... }
when: ref expired: { ... }
whenever: ref disconnected: { ... }
whenever: ref reconnected: { ... }
```
Distributed P2P applications

Urbiflock

WeScribble

PortalPong

WePong

WePoker

http://tinyurl.com/ambienttalkyoutube
Many times data needed to be shared...

def counter := isolate: {
    def val := 0;
    def incr(){ val := val + 1 };
    def decr(){ val := val - 1 };
    def value(){ val };
};

✅ increasing availability: operations can execute locally
✅ improving reliability: avoid single points of failure.

❌ providing some form of consistency is left to the application programmer.
Could we build replicated objects so that...

- developers can customize conflict resolution according to the application's needs

```c
Bayou_Write( update = { Insert, Meetings, 12/18/95, 10:00am, 60min, Project Meeting: Kevin),
   failure_check = {
      query = SELECT key FROM Meetings WHERE date = 12/18/95
      AND start < 11:00am AND end > 10:00am
      expected_result = EMPTY },
   mergeproc = {
      alternates = {12/18/95, 12:00pm};
      newupdate = {};
   FOREACH a IN alternates {
      # check if there would be a conflict
      IF NOT EMPTY
         SELECT key FROM Meetings WHERE date = a.date
         AND start + a.time = 60min AND end = a.time)
      CONTINUE;
      # no conflict, can schedule meeting at that time
      newupdate = { Insert, Meetings, a.date, a.time, 60min, Project Meeting: Kevin};
   };
   BREAK;
   IF (newupdate = {}) # no alternate is acceptable
      newupdate = { Insert, ErrorLog, 12/18/95, 10:00am, 60min, Project Meeting: Kevin}
   RETURN newupdate;
)
Could we build replicated objects so that ..

- developers can customize conflict resolution according to the application's needs
- without exposing them to merge procedures
Replicated Data Types (RDTs)

**Specification 6 State-based increment-only counter (vector version)**

1: payload integer[n] P
2: initial [0,0,...,0]
3: update increment ()
4: let g = myID()
5: P[g] := P[g] + 1
6: query value () : integer v
7: let v = \( \sum_i P[i] \)
8: compare (X, Y) : boolean b
9: let b = (\( \forall i \in [0,n-1] : X.P[i] \leq Y.P[i] \))
10: merge (X, Y) : payload Z
11: let \( \forall i \in [0,n-1] : Z.P[i] = \max(X.P[i], Y.P[i]) \)

Shapiro et al. 2011
Replicated Data Types (RDTs)

Shapiro et al. 2011

```plaintext
def statedBasedCounter := object:
    def vInc;
    def myId;

    def init(typeName, id, n) { .. };
    def increment() {
        def val := vInc.at(myId);
        vInc.atPut(myId, val +1);
    }
    def value() {
        def res := 0;
        vInc.each: { |val|
            res := res + val }
        res
    }
    def merge(senderVector) {
        def i := 0;
        vInc.each: { |a|
            def b := senderVector.get(i);
            vInc.atPut(i, Math.max(a, b));
            i := i +1};
        ...
    }
```
Replicated Data Types (RDTs)

**Specification 6 State-based increment-only counter (vector version)**

```
import CRDTModule.CRDTTrait;

def statedBasedCounter := object: {
    def vInc;
    def myId;

    def init(typeName,id,n) { .. };
    def increment() {
        def val := vInc.at(myId);
        vInc.atPut(myId, val +1);
    }
    def value() {
        def res := 0;
        vInc.each: { |val|
            res := res + val ;
        }
        res
    }
    def merge(senderVector) {
        def i := 0;
        vInc.each: { |a|
            def b := senderVector.get(i);
            vInc.atPut(i, Math.max(a,b));
            i := i + 1}
    }
    ...
}
```

**RDT Distribution aspects**

```
def CRDTTrait := object: {
    def typeName := defaultCRDT;
    def replicas := [];

    def sync(){
        self.broadcast(<-merge(self.serialize()));
    }

    def broadcast(msg) {
        self.replicas.each: { |farRef|
            farRef <+ msg
        }
    }

    def goOnline(){
        export: self as: (self.typeName);
        whenever: (self.typeName) discovered: {
            |farRef|
            self.replicas := self.replicas + [farRef];
        }
    }
}
```
Could we build replicated data types that...

- are application-specific?
- customize concurrency semantics to the application needs?
- support application invariants?
- are correct out-of-the-box?
- can be arbitrarily composed?
- can be applied to dynamic environments with memory and network constraints?
ECROs

Simplifying the development of application-specific RDTs

Explicitly Consistent Replicated Object (ECRO)

- General approach for building hybrid RDTs
  - Sequential data type
    - Distributed Specification
    - Replicated data type
  - Avoids unnecessary coordination

Fast when possible (EC)
consistent when needed (SC)
Explicitly Consistent Replicated Object (ECRO)

- General approach for building hybrid RDTs

**sequential data type**

```scala
case class AMSet[A](set: Set[A]) {
  def add(x: A) = new AMSet(set + x)
  def remove(x: A) = new AMSet(set - x)
  def contains(x: A) = set.contains(x)
}
```

**Distributed Specification**

Fast when possible (EC)  
consistent when needed (SC)
Implementing an Add-Wins Set

Sequential implementation in Scala

```scala
case class AWSet[V](set: Set[V]) {
  def add(x: V) =
    new AWSet(set + x)
  
  def remove(x: V) =
    new AWSet(set - x)
  
  def contains(x: V) =
    set.contains(x)
}
```
Implementing an Add-Wins Set

Sequential implementation in Scala

```scala
case class AWSSet[V](set: Set[V]) {
  def add(x: V) = 
    new AWSSet(set + x)

  def remove(x: V) = 
    new AWSSet(set - x)

  def contains(x: V) = 
    set.contains(x)
}
```

DSL for distributed specification

```scala
object AWSSet {
  // contains: V × State × Bool
  val contains: Relation = ...

  postcondition of add {
    (old: OldState, res: NewState) =>
    contains(x, res) /
    contains.copyExcept(old -> res, elem === x)
  }

  postcondition of remove {
    (old: OldState, res: NewState) =>
    not (contains(x, res)) /
    contains.copyExcept(old -> res, elem === x)
  }

  invariant on add {
    (_.: OldState, res: NewState) =>
    contains(x, res)
  }
}
```
Implementing an Add-Wins Set

Sequential implementation in Scala

```scala
case class AWSSet[V](set: Set[V]) {
  def add(x: V) =
    new AWSSet(set + x)

  def remove(x: V) =
    new AWSSet(set - x)

  def contains(x: V) =
    set.contains(x)
}
```

DSL for distributed specification
Building Geo-Distributed Apps, the ECRO Way

ECRO Data Type

Sequential implementation

```scala
case class AWSet[VJ](set: Set[VJ]) {
  def add(x: VJ) =
    new AWSet(set + x)
  def remove(x: VJ) =
    new AWSet(set - x)
  def contains(x: VJ) =
    set.contains(x)
}
```

DSL for distributed specification

Replicated Data Type

Translation

Ordana Analysis Tool

SMT Solver
Ordana: Statically Analyzes Distributed Specs

Derives information about:

1. Commutative methods
2. Conflicting methods

And finds:

3. Coordination-free solutions to conflicts
4. Fine-grained locks if no solution can be found
Serializing Operations: the ECRO Algorithm

- Replicas serialize operations locally
- strong convergence
- invariant preservation (i.e. safety)

Algorithm 1: ECRO replication algorithm main functions

1. \( \text{begin}(\text{ECRO}(\text{object})) \) // begin execution
2. \( \text{begin}(\text{restriction}(\text{object})) \) // acquire restrictions
3. \( \text{begin}(\text{commit}(\text{object})) \) // commit object's state
4. \( \text{end}(\text{ECRO}(\text{object})) \) // end execution
5. \( \text{end}(\text{restriction}(\text{object})) \) // release restrictions
6. \( \text{end}(\text{commit}(\text{object})) \) // end commit

Method call propagation:

- \( \text{Add-Wins Set Replica} \)
- \( \text{Object's State} \)
- \( \text{ECRO Replication Algorithm} \)

Machine 1

Replicated Objects

Add-Wins Set Replica

Object's State

\( \text{val set: } \{5, 12, 8, 17\} \)

ECRO Replication Algorithm

hb
co

Machine 2

Replicated Objects

Add-Wins Set Replica

Object's State

\( \text{val set: } \{5, 12, 8, 17\} \)

ECRO Replication Algorithm

hb
co

Diagram of method call propagation and object states on two machines.
Validation

Performance of ECROs vs PoR and RedBlue consistency

Well-known CRDTs
- Counter
- EW-Flag
- DW-Flag
- AW-Set
- RW-Set
- AW-Map
- RW-Map
- List

No CRDT
- Stack
- Queue

Application Specific
- RUBiS

Latency (in ms)
ECROs: Take Aways

• Augment sequential DT with distributed specification

• Static analysis is key to derive efficient RDTs
  • allows for informed decision at runtime

• But… separate specification
  • in FOL —> non-trivial, error-prone
  • subtle errors —> runtime anomalies
  • must evolve along with the code
How to ease the development of ECROs?

• High-level OOP language for sequential DTs

• Define concurrency semantics and invariants

• Fully compilable to SMT
  —> FOL specifications for free

• Synthesizes ECROs
The EFx language

---

**EFx**

- implementation
- concurrency contract

**Analyzer**

- EFx AST
- SMT-LIB code
- Ordana analysis
- RDT info

**Compiler**

- SMT plugin
- JS plugin
- Scala plugin
- ...
trait SetOps[V] {
  val set: Set[V]
  protected def copy(set: Set[V]): SetOps[V]

  def contains(elem: V) = this.set.contains(elem)
  def add(elem: V) = this.copy(this.set.add(elem))
  def remove(elem: V) = this.copy(this.set.remove(elem))
}

@replicated
class AWSet[V](set: Set[V]) extends SetOps[V] {
  protected def copy(set: Set[V]) =
    new AWSet(set)

  // add wins
  inv add(elem: V) {
    this.contains(elem)
  }
}
Remove-Wins Set in EFx

trait SetOps[V] {
  val set: Set[V]
  protected def copy(set: Set[V]): SetOps[V]

  def contains(elem: V) = this.set.contains(elem)
  def add(elem: V) = this.copy(this.set.add(elem))
  def remove(elem: V) = this.copy(this.set.remove(elem))
}

@replicated
class AWSerset[V](set: Set[V]) extends SetOps[V] {
  protected def copy(set: Set[V]) =
    new AWSerset(set)

  // add wins
  inv add(elem: V) {
    this.contains(elem)
  }
}

@replicated
class RWSet[V](set: Set[V]) extends SetOps[V] {
  protected def copy(set: Set[V]) =
    new RWSet(set)

  // remove wins
  inv remove(elem: V) {
    !this.contains(elem)
  }
}
## Validation: Portfolio of RDTs

<table>
<thead>
<tr>
<th>Data Type</th>
<th>LoC</th>
<th>C</th>
<th>M</th>
<th>Description and distributed semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>Supports increments and decrements.</td>
</tr>
<tr>
<td>EW-Flag</td>
<td>13</td>
<td>1</td>
<td>2</td>
<td>Flag that can be enabled and disabled. Enable wins over concurrent disable operations.</td>
</tr>
<tr>
<td>DW-Flag</td>
<td>13</td>
<td>1</td>
<td>2</td>
<td>Similar to EW-Flag but guarantees disable-wins semantics.</td>
</tr>
<tr>
<td>AW-Set</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>Set providing add-wins semantics for concurrent adds and removes of the same element.</td>
</tr>
<tr>
<td>RW-Set</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>Set providing remove-wins semantics.</td>
</tr>
<tr>
<td>LWW-Set</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>Set providing last-writer-wins semantics.</td>
</tr>
<tr>
<td>LWW-Array</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>Array providing last-writer-wins semantics for concurrent writes on the same index.</td>
</tr>
<tr>
<td>Sync-Array</td>
<td>24</td>
<td>1</td>
<td>1</td>
<td>Array with coordinated writes (locks index before writing).</td>
</tr>
<tr>
<td>AW-Map</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>Map with add-wins semantics for concurrent adds and removes of the same key, and last-writer-wins semantics for concurrent adds of the same key.</td>
</tr>
<tr>
<td>RW-Map</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>Similar to AW-Map but remove-wins semantics for concurrent adds and removes of the same key.</td>
</tr>
<tr>
<td>Stack</td>
<td>14</td>
<td>1</td>
<td>2</td>
<td>Stack allowing push, pop, and top operations. Push operations execute optimistically and are totally ordered. Pop operations are coordinated in order not to pop more elements than there are on the stack.</td>
</tr>
<tr>
<td>Queue</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>Enqueue operations run optimistically and are totally ordered. Dequeue operations are coordinated to avoid dequeuing more elements than there are in the queue.</td>
</tr>
<tr>
<td>VotingGame</td>
<td>53</td>
<td>3</td>
<td>2</td>
<td>A distributed voting game inspired by contemporary tv-shows [Cet+14].</td>
</tr>
<tr>
<td>SmallBank</td>
<td>90</td>
<td>2</td>
<td>4</td>
<td>Banking application corresponding to the SmallBank benchmark [Alo+08].</td>
</tr>
<tr>
<td>RUBS</td>
<td>87</td>
<td>2</td>
<td>6</td>
<td>Auction system similar to the RUBS benchmark [EJ09].</td>
</tr>
<tr>
<td>Airline</td>
<td>285</td>
<td>9</td>
<td>9</td>
<td>An airline reservation system inspired by Acme Air [TS].</td>
</tr>
</tbody>
</table>
VeriFx

Correct replicated data types for the masses
The VeriFx Language

- High-level OOP language with extensive functional collections
  - maps, sets, vectors, etc.

- Features a novel proof construct
  - used by programmers
  - describe application-specific correctness properties

- Also fully compilable to SMT -> Automated proof verification

http://verifx.org/
VeriFx’s Iterative Workflow for developing RDTs

1. Design RDT
2. Implement RDT in VeriFx
3. Automated verification
   - correct? yes -> Transpile
   - correct? no -> Modify RDT implementation
4. Interpret counterexample
5. Deploy in system

http://verifx.org/
Supporting development of distributed systems goes beyond providing novel programming models
Tooling is essential!
Reasoning about distributed events..

Generate and receive application requests

\[
\text{obj} <- \text{msg}(<\text{arg}>)
\]
\[
\text{def} \ \text{msg}(\text{param}) \ { \ ... \ }
\]

Follow-up on outstanding requests

\[
\text{when: future becomes:} \ { \ | \text{result|} \ ... \ }
\]

React to services appearing and disappearing

\[
\text{when: type discovered:} \ { \ |\text{ref|} \ ... \ }
\]
\[
\text{whenever: type discovered:} \ { \ |\text{ref|} \ ... \ }
\]

React to references disconnecting, reconnecting, and expiring

\[
\text{when: ref disconnected:} \ { \ ... \ }
\]
\[
\text{when: ref reconnected:} \ { \ ... \ }
\]
\[
\text{when: ref expired:} \ { \ ... \ }
\]
\[
\text{whenever: ref disconnected:} \ { \ ... \ }
\]
\[
\text{whenever: ref reconnected:} \ { \ ... \ }
\]

REMEO Breakpoint Catalog

Message resolution breakpoint

shoppingSession<-go()@FutureMessage

resolve(value)
REME-D Stepping

Step Over

**Step Into**

Step Return

Step Until

buyerP

goto

inventory

partInStock

credit bureau

checkCredit

shipper

canDeliver
Pre-experimental User Study

**Goal:** How real users perceive and value the features of an ambient-oriented (AmOP) debugger.

- One-group pretest-posttest quasi-experiment design.
- 22 participants.
Pre-experimental User Study: Take Home Message

• Users value REME-D as tool to make AmOP programming in AmbientTalk easier.

• REME-D supports expected features for an ambient-oriented debugger.

• Impact of UI and visualisations.
Could we build debugging support that...

• deals with non-determinism inherent to distributed systems?

• can be applied to different concurrency models?

• features advanced visualisations for the event-based nature of distributed systems?

• is probe-effect free?

• deals with big amounts of data?

• can be used in environments with memory and network constraints?
Big Data Processing

Long Running
Due to the high volume of data they have to analyze

Distributed
They remotely execute on clusters, which slows down the debugging cycle
Bugs in Big Data Processing Applications

37% of Reported Errors
In cloud Big Data processing services are attributed to developer errors [Zhou et al. 2015]

Code Defect
Explicit errors inserted by developers

Operation Fault
Common operational mistakes, e.g., file renaming

Misuse
A configuration error, e.g., using a wrong library version
Could we build a debugger so that...

**Online Debugging**
Debug the system when the bug happens

**Global View**
Centralised debugging of the distributed system

**Isolation**
Debug the system without interfering with its execution

**Updates of the Running System**
Deploy code-fixes without restarting the whole distributed system
Out-of-Place Debugging

- Developer Machine
- Cluster

Master

Worker

Debug Session
Exception

Debug Session
Exception

Avoid Replays
Domain-Specific Debugging
Debugging Session
Captures the execution state through the call-stack

Remove Framework Frames
Reduce the amount of data to be transferred

Include the **event-inducing record**
I.e., the record that was being processed when the debugging event (breakpoint or error) happened.

Include the partition of the **event-inducing record**
The partition of data that was being processed when the debugging event happened, that includes the event-inducing record
Distributed Live Code Updates

Developer Machine | Cluster

Exception
Commit
Patch

Patch

Master

Worker

Worker
**Debug Single Record**
Select which debugging event to debug starting from the event-inducing record, including its partition

**Debug on Virtual Partitions**
Including all of the event-inducing records, or a merge of all their partitions
Spa: a Live Debugger for Spark

**Classic stepping operations**
Typical of online debuggers

**Dedicated stepping operations**
Tailored to Spark-like computations

https://www.youtube.com/watch?v=GpipdhVxYq0
Event-based Out-of-place Debugging

Practical Online Debugging of Internet of Things applications

Out of Place Debugging for Internet of Things
Non-transferable resources

Domain-Specific Debugging
In Conclusion

- Distributed systems are varied, successful and widespread.
- They are still challenging to design and implement.
- It is essential to explore novel programming abstractions in tandem with software tools tailored to modern concurrent and distributed software.
Thanks to DisCo & collaborators!

<table>
<thead>
<tr>
<th>Dominik Aumayr</th>
<th>Aäron Munsters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim Bauwens</td>
<td>Florian Myter</td>
</tr>
<tr>
<td>Clément Béra</td>
<td>Isaac Nyabisa Oteyo</td>
</tr>
<tr>
<td>Dina Borrego</td>
<td>Guillermo Polito</td>
</tr>
<tr>
<td>Kevin De Porre</td>
<td>Nuno Preguiça</td>
</tr>
<tr>
<td>Carla Ferreira</td>
<td>Carlos Rojas Castillo</td>
</tr>
<tr>
<td>Robert Gurdeep Singh</td>
<td>Christophe Scholliers</td>
</tr>
<tr>
<td>Tom Lauwaerts</td>
<td>Angel Luis Scull Pupo</td>
</tr>
<tr>
<td>Stefan Marr</td>
<td>Carmen Torres Lopez</td>
</tr>
<tr>
<td>Matteo Marra</td>
<td>…</td>
</tr>
<tr>
<td>Hanspeter Mössenböck</td>
<td></td>
</tr>
</tbody>
</table>

✉️ egonzale@vub.be  🐦 @elisagboix

https://soft.vub.ac.be/disco/