CO2 honesty checker

Security Foundations

atzeinicola@gmail.com

Topics

- CO_2 Java API - part 2
- culpability, reputation, honesty
- CO_2 honesty checker

CO_2 Java API - part 2

What we have learned

- create a connection to the middleware
- define and advertise contracts
- retract a contract
- retrieve a session
- send and receive messages
- check if the session is ended (with success or not)
- check if I am on duty
- check if I am culpable
- what are contexts and what is their purpose

Sending messages

A participant can send messages to the other endpoint specifing an action and a optional value

```
session.send("a")
session.send("a", "string value")
session.send("a", 42)
```

The other participant can receive a message and get its value

```
msg = session.waitForReceive()
msg.getLabel()
 => "a"
msg.getStringValue()*
msg.getIntegerValue()*
```

* it throws a `ContractException` if the type is incorrect

New send API

By sending a message, you can become culpable if you perform the wrong action (i.e. an action not allowed by your contract in this moment)

```
session.sendIfAllowed("a")
session.sendIfAllowed("a", "string value")
session.sendIfAllowed("a", 42)
```

If the action `a` is not allowed by the contract, the method blocks the execution. However, if the action will be permitted afterwards, then the method is released and the action is sent.

Example

```
contract = "!foo . !bar"
...
①new Thread({session.sendIfAllowed("bar")}).start() *
②session.sendIfAllowed("foo")
```

At point ①, the program starts a new thread that executes `session.sendIfAllowed("bar")` in parallel. The action `bar` is not allowed, so the thread will block.

At point ②, the action `foo` is sent, so the action `bar` become allowed and the thread can terminate.

* the lambda expression `{...}` only works in groovysh, please use `()->{...}` in Java 8

Wait for receive

There is another way to receive a message. You can specify the labels you are waiting for, ignoring all the other ones, as follows

```
msg = session.waitForReceive("a", "b", "c")
msg = session.waitForReceive(10_000, "a", "b", "c")


switch (msg.getLabel()) {

 case "a": ...
 case "b": ...
 case "c": ...
}
```

Culpability and honesty

A participant is culpable if it does not fulfill its contract obligations.

We say that a participant is honest, if it can always fulfil its obligations. An honest participant can not be culpable in any session, even when interacting with a malicious user.

Reputation

The middleware associates a reputation to each participant.

```
CO2ServerConnection.getReputation()
```

Upon detection of a contract violation, the middleware punishes the culprit, by suitably decreasing its reputation.

This is a measure of the trustworthiness of a participant in its past interactions: the lower is the reputation, the lower is the probability of being able to establish new sessions with it.

Example

```
c = "!pay . ( (?ok . !card) & ?no )"

Session<TST> s = new TST(c).toPrivate(conn).tell().waitForSession();
s.send("pay");

① Message msg = s.waitForReceive("yes", "no");

if (msg.getLabel().equals("ok")) {
  s1.send("card");
}
```

At point ①, we typed `yes` instead of `ok`. If the other participant sends us `no`, we are lucky and we respect our contract. On the contrary, if it sends `ok` (so respecting its dual contract), we become culpable because we will never send the action `card`.

Example - multiple sessions

```
c1 = "?hello . ( !pay + !abort )"
c2 = "!pay . ( ?ok & ?no )"

Session<TST> s1 = new TST(c1).toPrivate(conn).tell().waitForSession();
s1.waitForReceive("hello");

Session<TST> s2 = new TST(c2).toPrivate(conn).tell().waitForSession();
s2.sendIfAllowed("pay");

Message msg = s2.waitForReceive("ok", "no");

if (msg.getLabel().equals("ok")) {
  s1.sendIfAllowed("pay");
}
else {
  s1.sendIfAllowed("abort");
}
```

What happens if `s2` is not established?

What happens if `s2.waitForReceive()` never returns?

DISHONEST!

Example - multiple sessions (2)

```
c1 = "?hello . ( !pay + !abort )"
c2 = "!pay . ( ?ok & ?no )"

Session<TST> s1 = new TST(c1).toPrivate(conn).tell().waitForSession();
s1.waitForReceive("hello");

Public<TST> pbl2 = new TST(c2).toPrivate(conn).tell(10_000);

try {
  Session<TST> s2 = pbl2.waitForSession();   // can throws ContractExpiredException
  s2.sendIfAllowed("pay");

  Message msg = s2.waitForReceive("ok", "no");

  if (msg.getLabel().equals("ok"))
    s1.sendIfAllowed("pay");
  else
    s1.sendIfAllowed("abort");

} catch (ContractExpiredException e) {
  s1.sendIfAllowed("abort");
}
```

DISHONEST!

Example - multiple sessions (3)

```
Session<TST> s1 = new TST(c1).toPrivate(conn).tell().waitForSession();
s1.waitForReceive("hello");

Public<TST> pbl2 = new TST(c2).toPrivate(conn).tell(10_000);

try {
  Session<TST> s2 = pbl2.waitForSession();   // can throws ContractExpiredException
  s2.sendIfAllowed("pay");

  try {
    Message msg =
       s2.waitForReceive(10_000, "ok", "no"); // can throws TimeExpiredException

    if (msg.getLabel().equals("ok"))
      s1.sendIfAllowed("pay");
    else
      s1.sendIfAllowed("abort");

  } catch (TimeExpiredException e) {
    s1.sendIfAllowed("abort");
  }
} catch (ContractExpiredException e) {
  s1.sendIfAllowed("abort");
}
```

DISHONEST!

Example - multiple sessions (4)

```
Session<TST> s1 = new TST(c1).toPrivate(conn).tell().waitForSession();
s1.waitForReceive("hello");

Public<TST> pbl2 = new TST(c2).toPrivate(conn).tell(10_000);

try {
  Session<TST> s2 = pbl2.waitForSession();   // can throws ContractExpiredException
  s2.sendIfAllowed("pay");

  try {
    Message msg =
       s2.waitForReceive(10_000, "ok", "no"); // can throws TimeExpiredException

    if (msg.getLabel().equals("ok"))
      s1.sendIfAllowed("pay");
    else
      s1.sendIfAllowed("abort");

  } catch (TimeExpiredException e) {
    s1.sendIfAllowed("abort"); s2.waitForReceive("ok", "no");
  }
} catch (ContractExpiredException e) {
  s1.sendIfAllowed("abort");
}
```

HONEST!

CO_2 honesty checker

CO_2 honesty checker

CO_2 honesty checker is a Java tool/library you can use to verify if your program is honest.

Using timed session types, the tool does not consider the time-guards to check the honesty, so a honest program can become dishonest when considering time (and viceversa).

This is due to the fact that we cannot predict how much time a program can take to perform an action.

Example

```
TST contract = new TST("!a{x<5} + !b{x<5}");
...
session = ...
Thread.sleep(10_000);
session.send("a");
```

The program is honest when considering its untimed version. However the send will fail at runtime because the time is expired.

Considering the time, the program is dishonest.

Example

```
TST contract = new TST("?a{x<5} + ?b{x<5}");
...
session = ...
session.waitForReceive(5_000, "a", "b");
```

The program is dishonest when considering its untimed version. However the program works as expected at runtime.

Considering the time, the program is honest.

Participant class

The abstract class Participant models a participant who wants to interact with the middleware. It defines the constructor as

```
Participant.<init>(String username, String password)
```

Participant also implements Runnable, so you must override the `run` method. All the code into `run` defines the behaviour of your program and it is used to verify the honesty.

Participant class

The class Participant provides some methods to tell contracts

```
Participant.tell(Contract c) : Public<TST>
Participant.tellAndWait(Contract c) : Session<TST>
```

The Contract class allows to define contracts using some classes instead of a plain String. This is required in order to verify the honesty of a program.

It also provides the method `parallel(Runnable r)` to execute asynchronous code. You can use it as follows

```
parallel(() -> {
  // some code
});
```

Participant example

```
class Alice extends Participant {
  Contract c = ...

  public run() {
    Session<TST> session = tellAndWait(c);

    parallel(() -> {
      session.waitForReceive("y", "z");
    })
    session.sendIfAllowed("a");
  }
}
```

Honesty verification

You can check the honesty of a class that extends Participant, e.g.

```
Alice extends Participant { ... }
aliceInstance = new Alice(arg1, arg2);
```

as follows

```
HonestyChecker.isHonest(Alice.class, arg1, arg2);
HonestyChecker.isHonest(aliceInstance);
```

These methods return an enum HonestyResult:

- HonestyResult.HONEST (untimed version)
- HonestyResult.DISHONEST (untimed version)
- HonestyResult.UNKNOWN

Proof of presence

Provide the proof

In order to receive your token, you must implement an honest program and send it to a service that will validate it. You must send an abstract representation of your program, obtained using the co2-honesty-checker tool.

If your program is honest, you will receive the token.

Which program you need to implement and send? The program that you will use to obtain the token! (yes, the program will send itself)

Lab context

I have updated the context named lab as follows:

Action label	Verification link	Type
matricola	http://co2.unica.it/verifier/lab_co2_2015_verification_link.php*	int
proc	http://co2.unica.it/verifier/true.php	string
honest	http://co2.unica.it/verifier/true.php	string
dishonest	http://co2.unica.it/verifier/true.php	-
unknown	http://co2.unica.it/verifier/true.php	-

* please verify that the link
http://co2.unica.it/verifier/lab_co2_2015_verification_link.php?value=<matricola> returns true

Provide the proof

Download the class Template.java from moodle.

The class already defines the contract:

```
!matricola{x<60;x} . !proc{x<60;x} . (
  ?honest{x<60}
  & ?dishonest{x<60}
  & ?unknown{x<60}
)
```

You have to complete the `run` method with the behaviour defined by the contract.

Provide the proof

- get a session by using `Participant.tellAndWait(Contract)`
- send your id (action `matricola`) *
- send the serialized process (action `proc`) *
- wait for receive the expected actions by contract (`honest`, `dishonest` and `unknown`) **

* you have to use `Session.sendIfAllowed(...)`
** you have to use `Session.waitForReceive(...)`

Solution

```
Session<TST> session = tellAndWait(contract);

session.sendIfAllowed("matricola", matricola);
session.sendIfAllowed("proc", processSerialized);

Message msg = session.waitForReceive(
 "honest",
 "dishonest",
 "unknown");

if (msg.getLabel().equals("honest"))
 System.out.println(msg.getStringValue());// it print your token
```