Modelling, Verifying and Testing the Contract Automata Runtime Environment with UPPAAL

Davide Basile

(COORDINATION 2024)

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

Formal Methods and Tools Lab

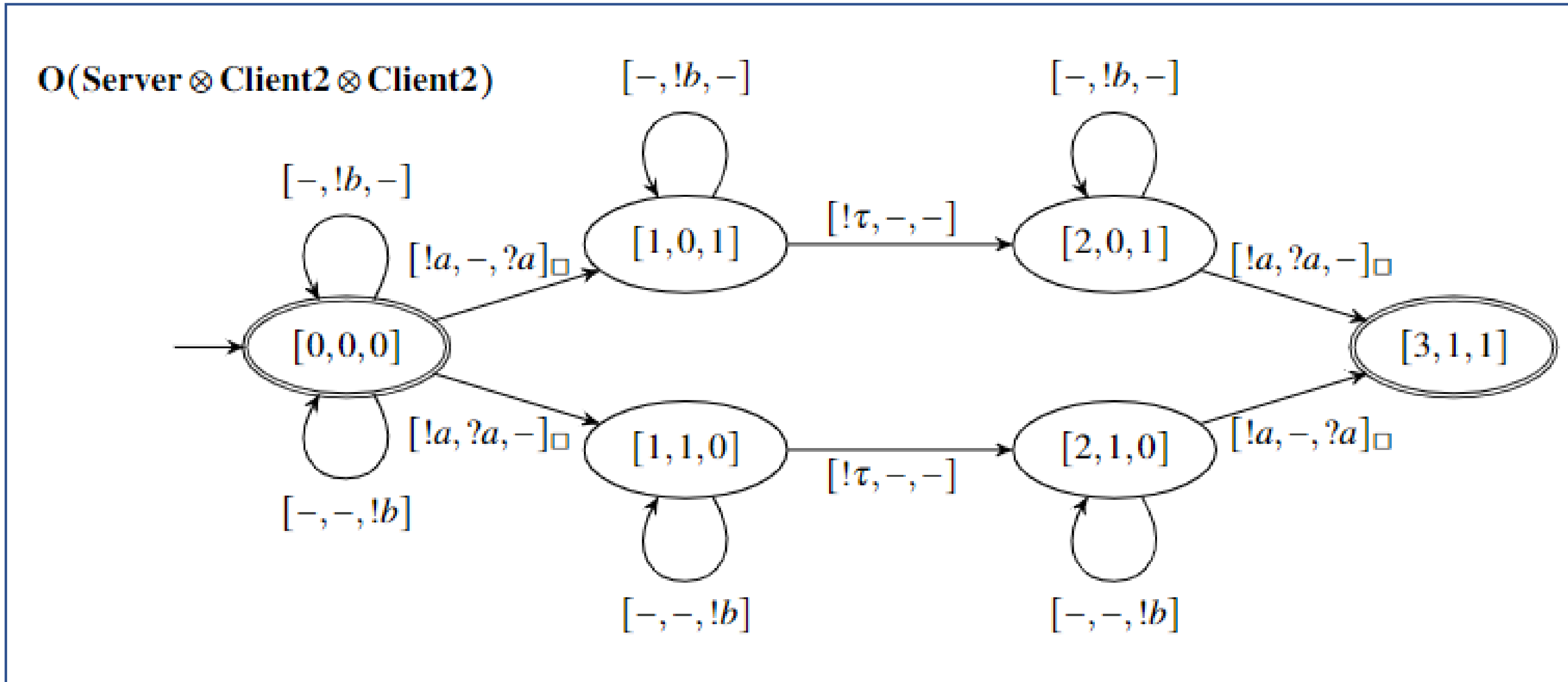
Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

Overview

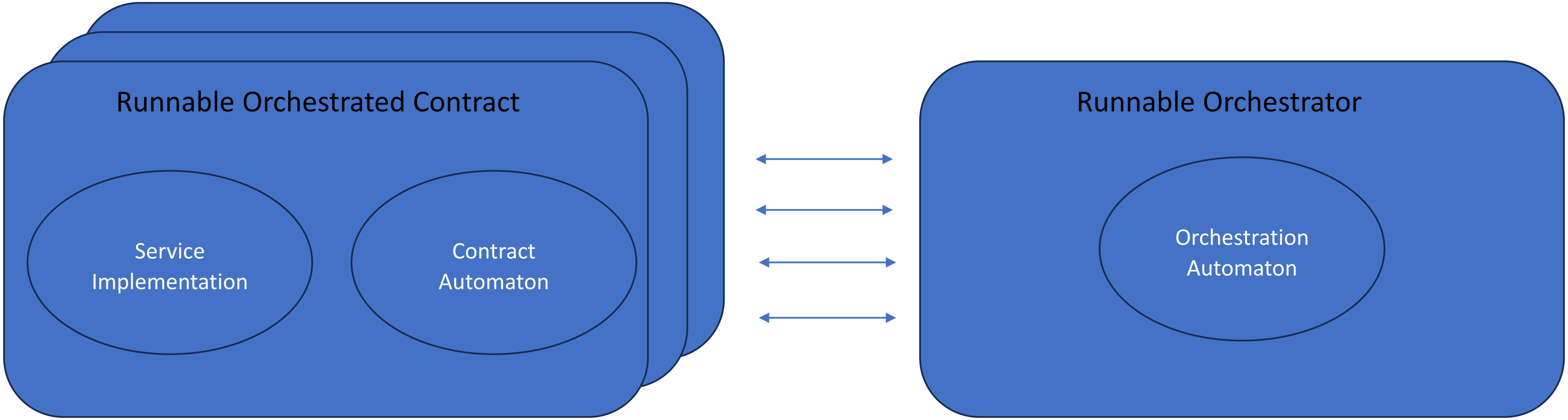
- Background: Contract Automata Runtime Environment, UPPAAL
- Formal model: abstractions, adequacy
- Formal analysis:
  - parameters tuning, statistical / exhaustive model checking
- Conclusion

Contract Automata

- Contract automata are FSA enhanced with:
  - Partitioned alphabet of actions:
    - offers !a (A^o) and requests ?a (A^r)
    - special idle action (- not in $A^o \cup A^r$)
  - rank : the number of services in the contract
  - States are list of basic states
  - Labels are list of actions and are constrained to be:
    - offers: (-, -, -, !a)
    - requests: (-, ?a, -, -)
    - matches: (-, ?a, -, !a)
      (only between two)
  - size(list) = rank
  - Orchestrator abstracted away



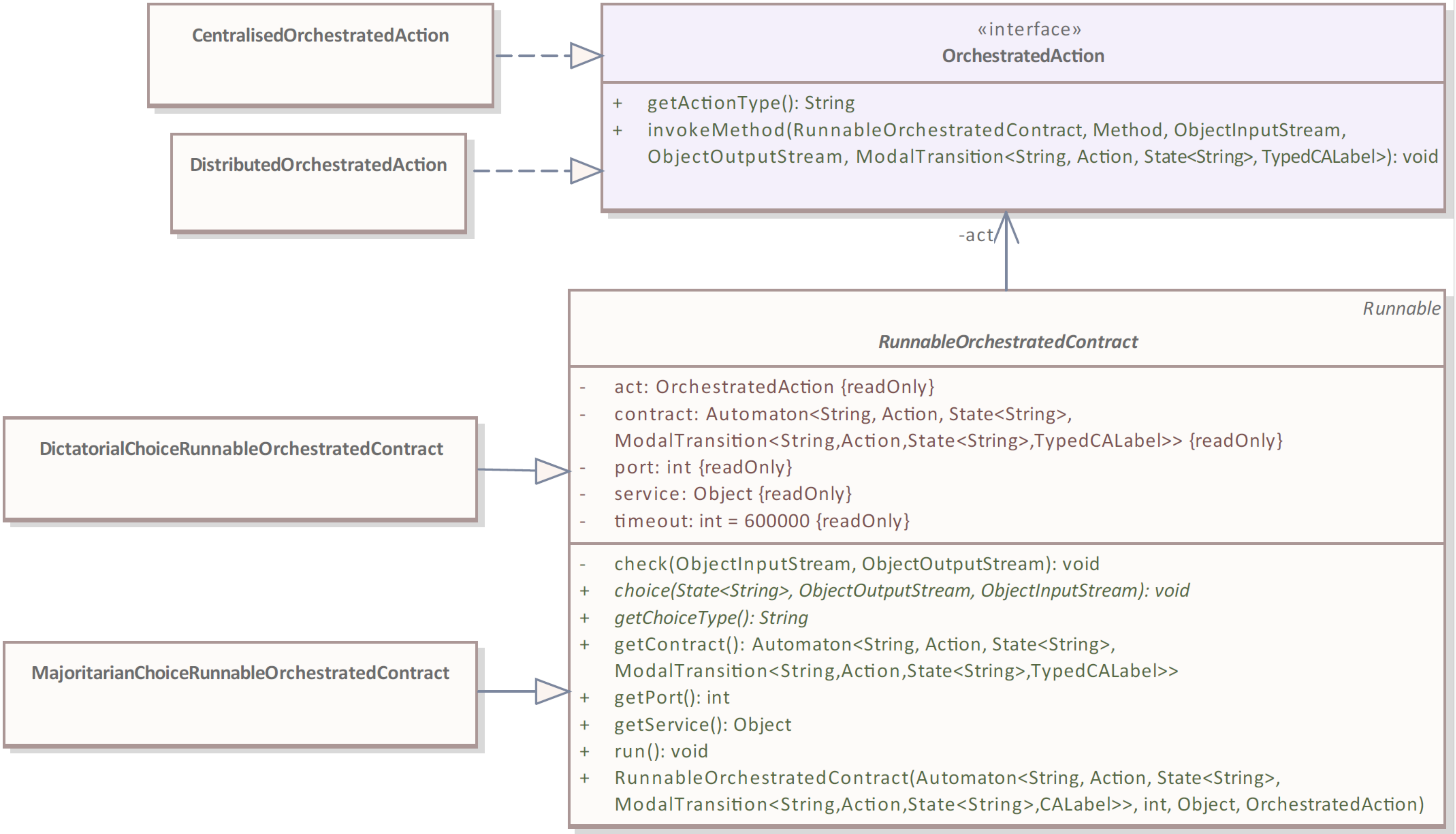
Contract Automata Runtime Environment



Basile, D. and ter Beek, M.H. A runtime environment for contract automata. In *FM 2023*

https://github.com/contractautomataproject/CARE
https://github.com/contractautomataproject/CARE/tree/master/src/spec/uppaal

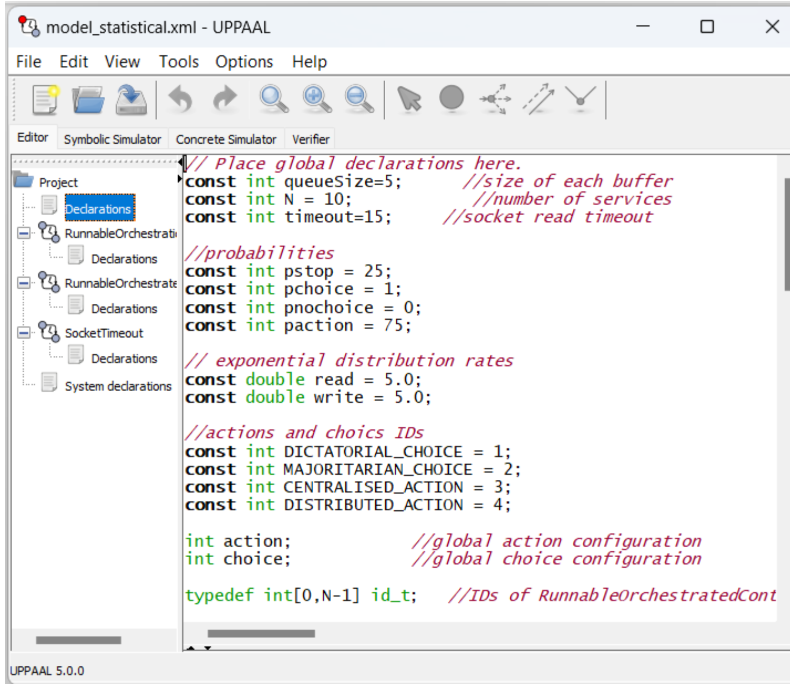
UPPAAL formally verified



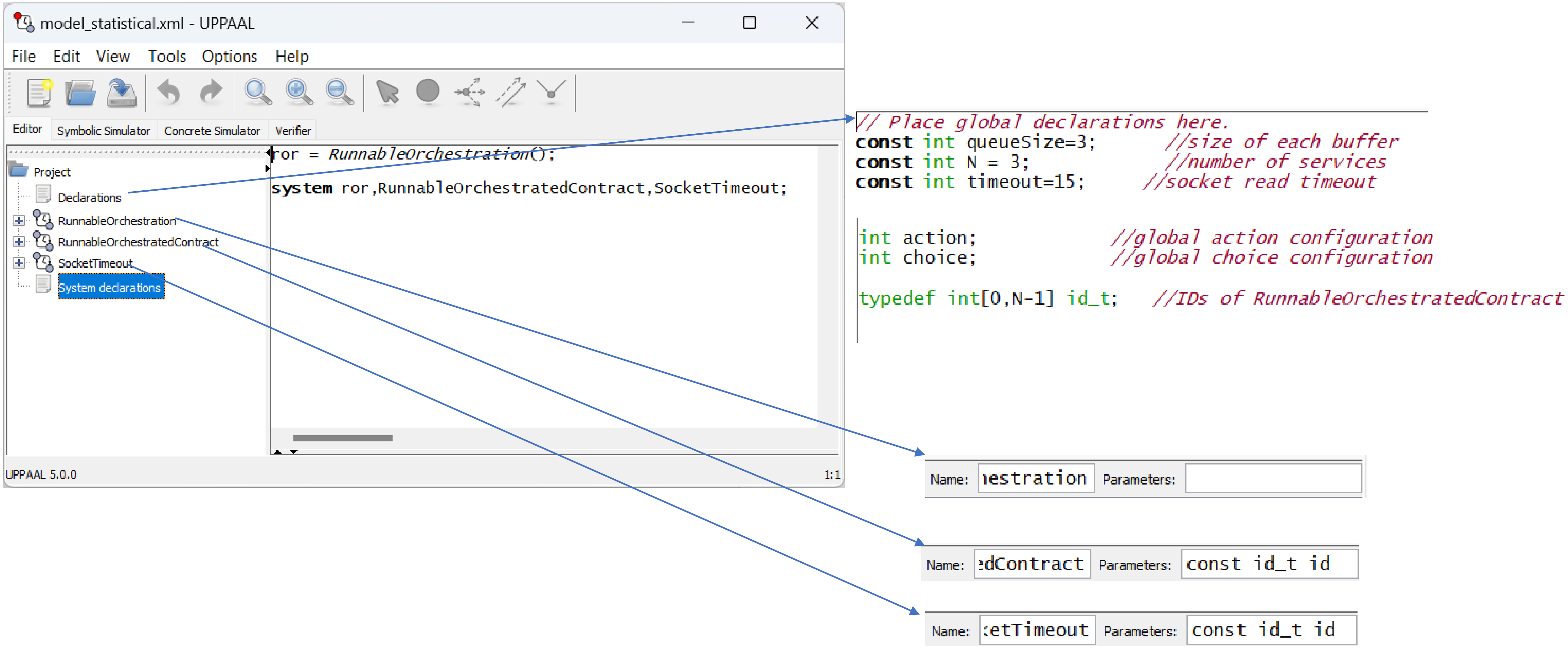
CentralisedOrchestratedAction
DistributedOrchestratedAction
«interface»
OrchestratedAction
+ getActionType(): String
+ invokeMethod(RunnableOrchestratedContract, Method, ObjectInputStream, ObjectOutputStream, ModalTransition<String, Action, State<String>, TypedCALabel>): void
-act
Runnable
RunnableOrchestratedContract
- act: OrchestratedAction {readOnly}
- contract: Automaton<String, Action, State<String>, ModalTransition<String,Action,State<String>,TypedCALabel>> {readOnly}
- port: int {readOnly}
- service: Object {readOnly}
- timeout: int = 600000 {readOnly}
- check(ObjectInputStream, ObjectOutputStream): void
+ choice(State<String>, ObjectOutputStream, ObjectInputStream): void
+ getChoiceType(): String
+ getContract(): Automaton<String, Action, State<String>, ModalTransition<String,Action,State<String>,TypedCALabel>>
+ getPort(): int
+ getService(): Object
+ run(): void
+ RunnableOrchestratedContract(Automaton<String, Action, State<String>, ModalTransition<String,Action,State<String>,CALabel>>, int, Object, OrchestratedAction)
DictatorialChoiceRunnableOrchestratedContract
MajoritarianChoiceRunnableOrchestratedContract

UPPAAL

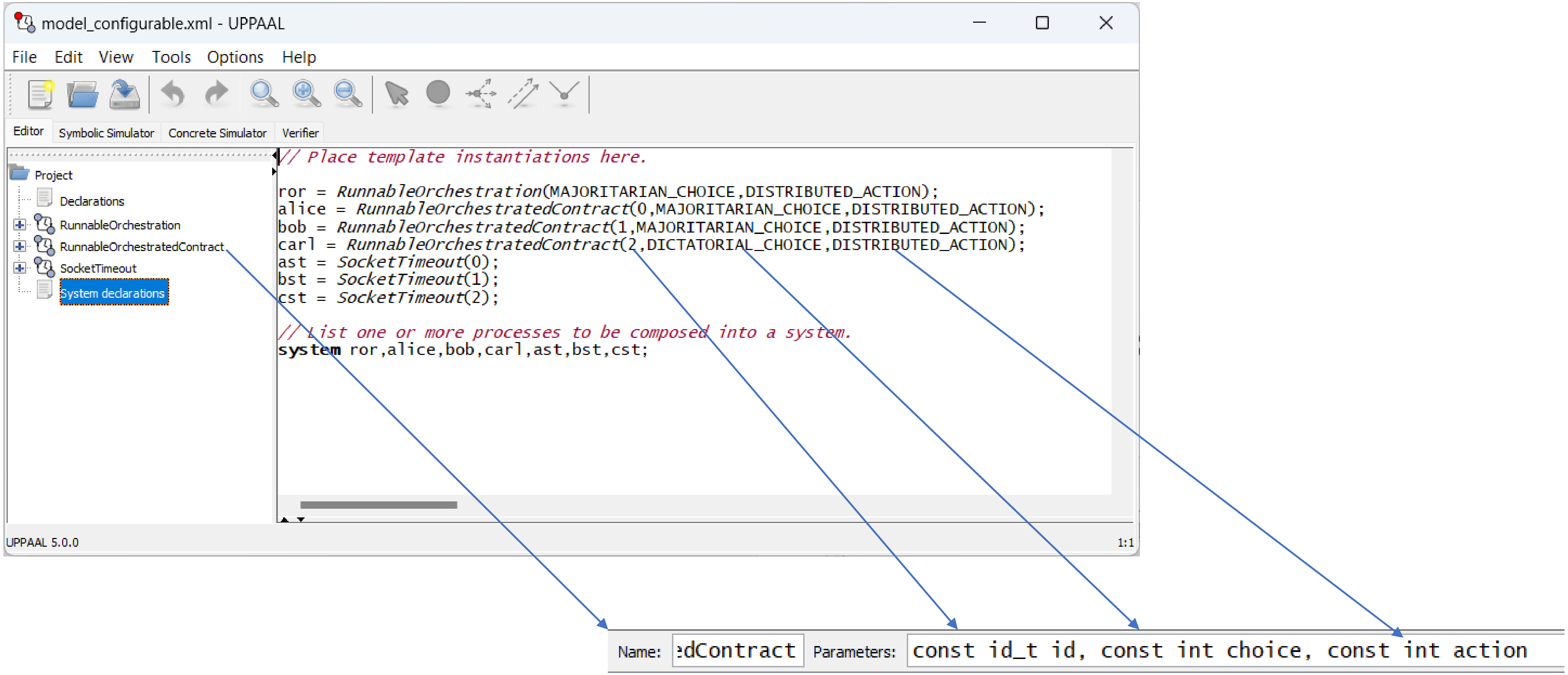
- Toolbox for the verification of real-time systems
  - Dialect of stochastic priced timed automata (probabilistic choices, probability distributions for delays)
  - Communications: broadcast channels, shared variables
  - Exhaustive model checking of a dialect of CTL properties
- Statistical model checking:
  - statistically estimate the probability of a formula
  - to hold by running a sufficient number of simulations, based on parameters (precision, confidence)
  - Independent of the size of the state-space
- Templates, Test generation, Simulation, etc…



Formal model: Network of UPPAAL Automata



Formal model: Network of UPPAAL Automata



Formal Model: Java TCP/IP Sockets

- Asynchronous with FIFO buffers, blocking

Global Declarations

```
int orc2services[N][queueSize];
int services2orc[N][queueSize];


//signals IDs
const int ORC_CHECK=1;
const int ORC_STOP=2;
const int ORC_CHOICE=3;
const int CHOICE_STOP=4;
const int ACK=5;
const int ERROR=6;
const int SKIP=7;
const int CHOICES=8;
const int SERVICE_CHOICE=9;
const int ACTION=10;
const int REQUEST=11;
const int OFFER=12;
const int TYPEOFFER=13;
const int TYPEMATCH=14;
const int PORT=15;
const int NOPAYLOAD=16;
const int ADDRESS=17;
```

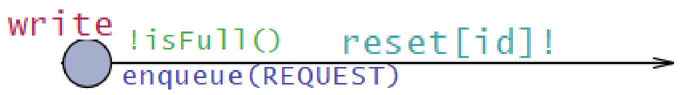
Runnable Orchestrated Contract

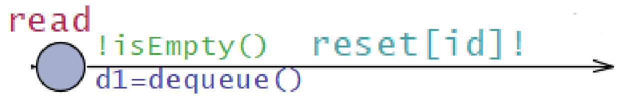
```
void enqueue(int ide_sig)
{
    int i;
    int s=ide_sig;
    for (i:=0;i<queueSize-1;i++)
    {
        services2orc[id][i]:=services2orc[id][i+1];
    }
    services2orc[id][queueSize-1]:=s;

}

int dequeue()
{
    int i;
    for (i:=0;i<queueSize;i++)
    {
        if (orc2services[id][i]!=nil)
        {
            int e=orc2services[id][i];
            orc2services[id][i]=nil;
            return e;
        }
    }
    return nil;
}
```

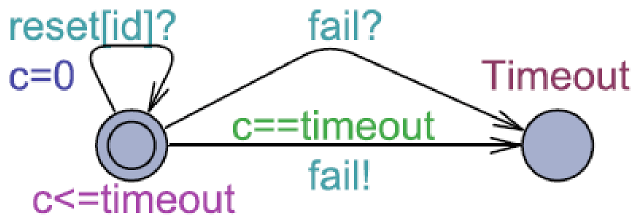
Runnable Orchestrator

```
void enqueue(int id,int ide_sig)
{
    int i;
    int s=ide_sig;
    for (i:=0;i<queueSize-1;i++)
    {
        orc2services[id][i]:=orc2services[id][i+1];
    }
    orc2services[id][queueSize-1]:=s;
}

int dequeue(int id)
{
    int i;
    for (i:=0;i<queueSize;i++)
    {
        if (services2orc[id][i]!=nil)
        {
            int e=services2orc[id][i];
            services2orc[id][i]=nil;
            return e;
        }
    }
    return nil;
}
```

Formal Model: Java TCP/IP Sockets

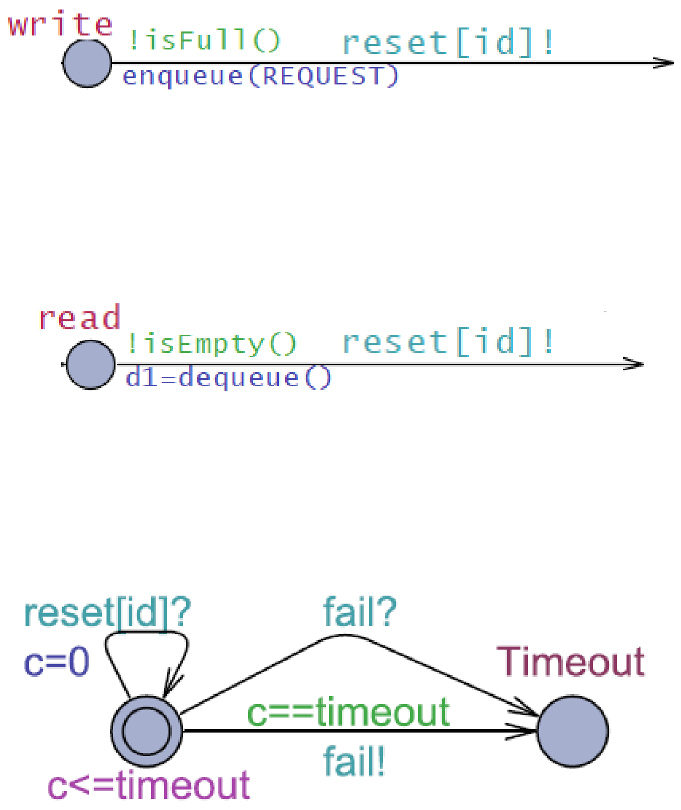




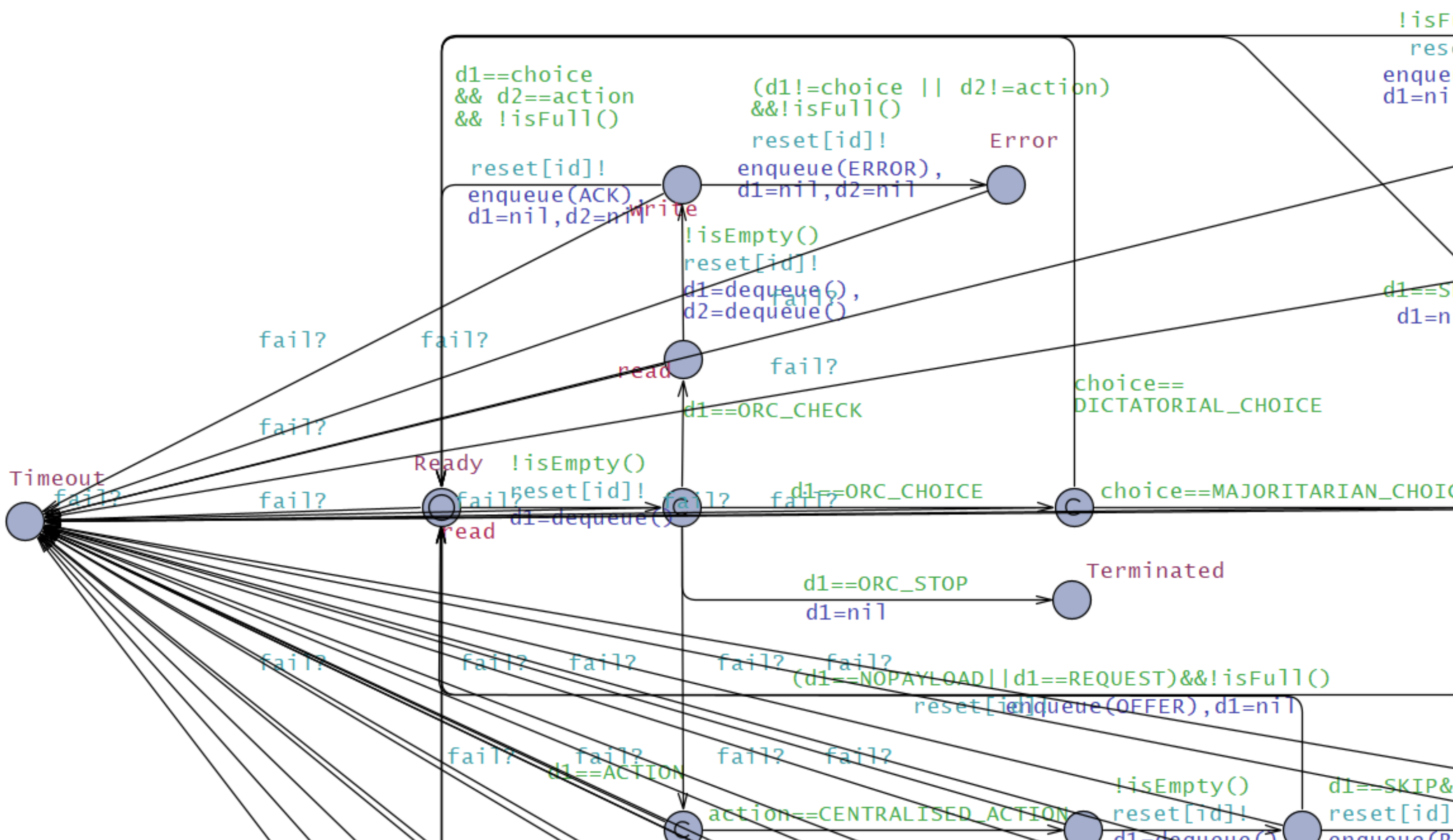


- Source locations: neither urgent nor committed
  - otherwise, there could be deadlocks
- Unbounded delays: timeout model

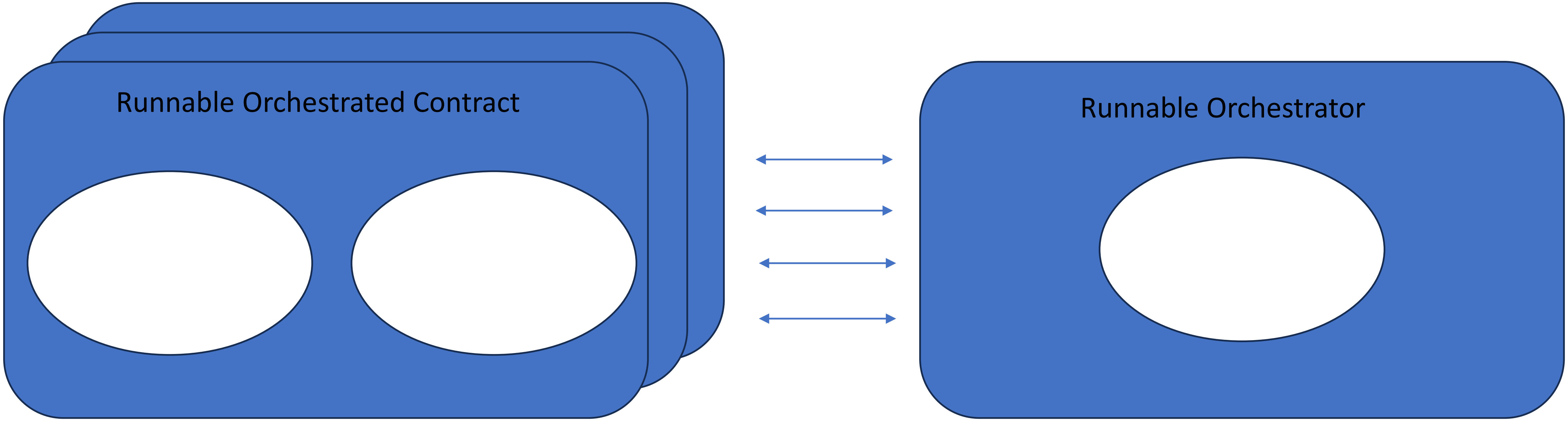
Formal Model: Java TCP/IP Sockets



- Source locations: neither urgent nor committed
  - otherwise, there could be deadlocks
- Unbounded delays: timeout model

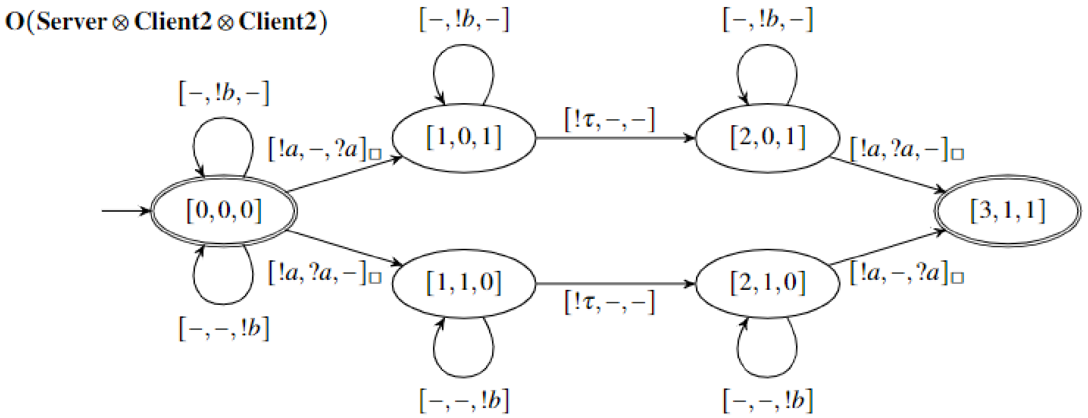


CARE Model: abstractions

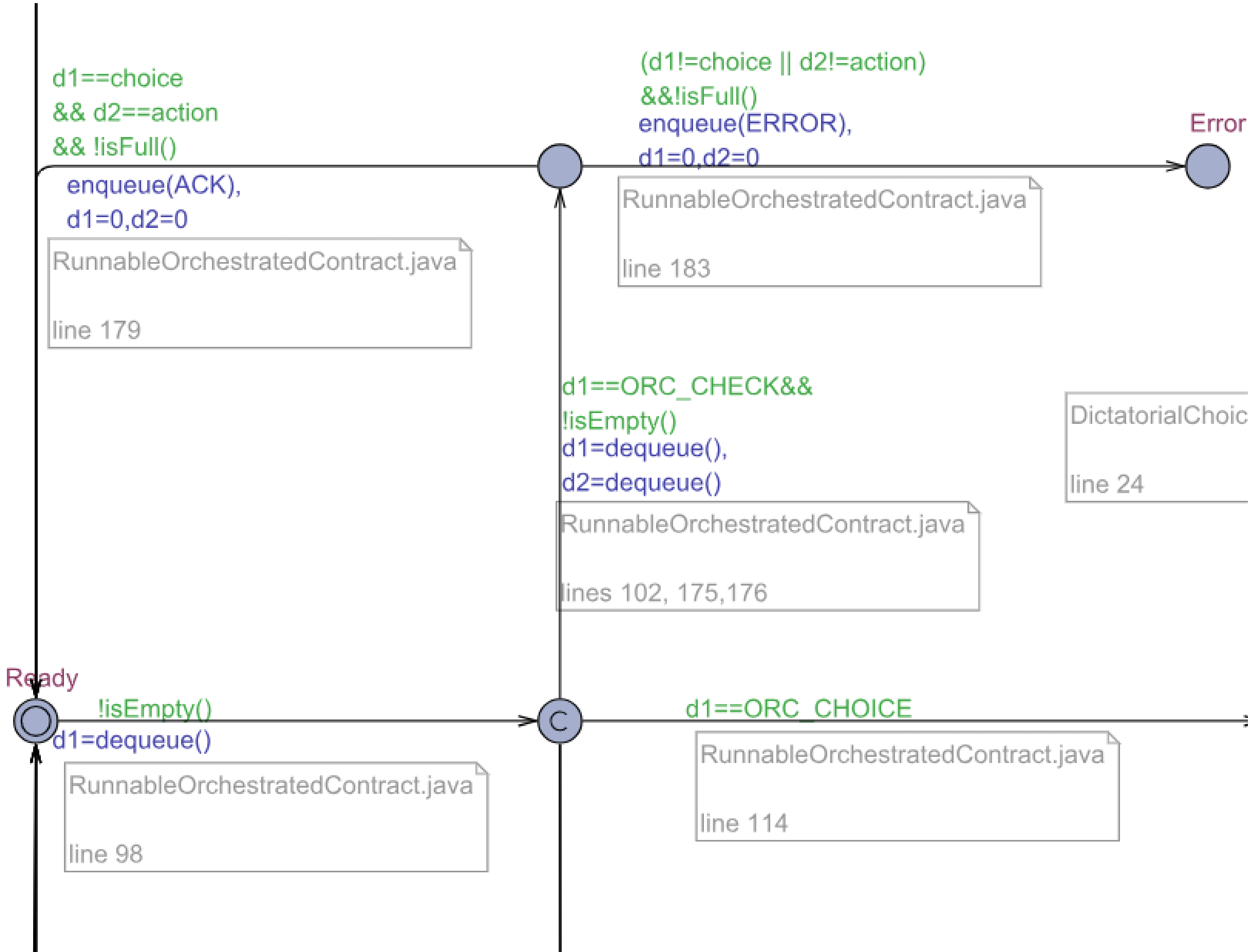


Other abstracted aspects:

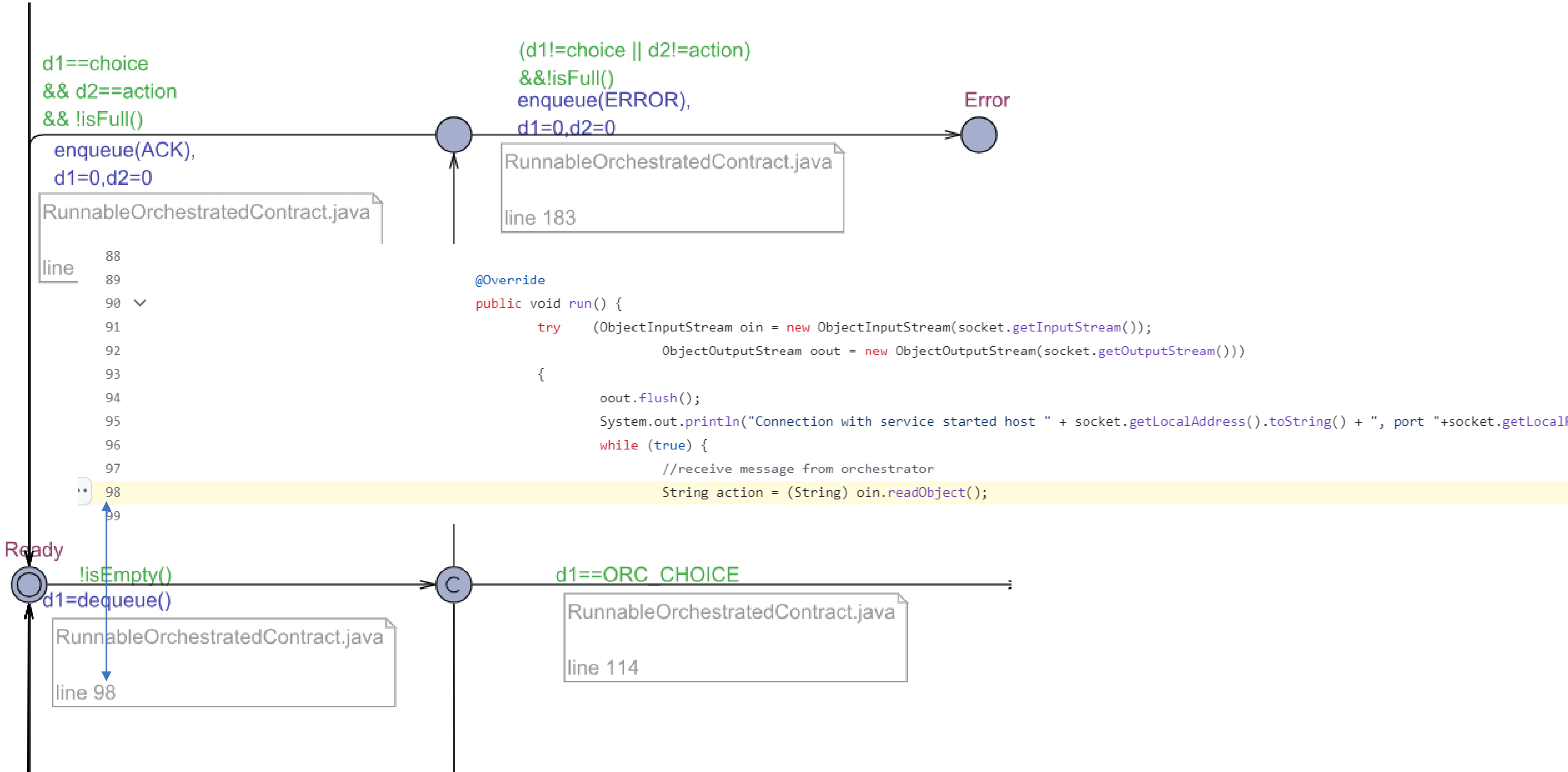
- payload of communications,
- conditionals,
- match/offer



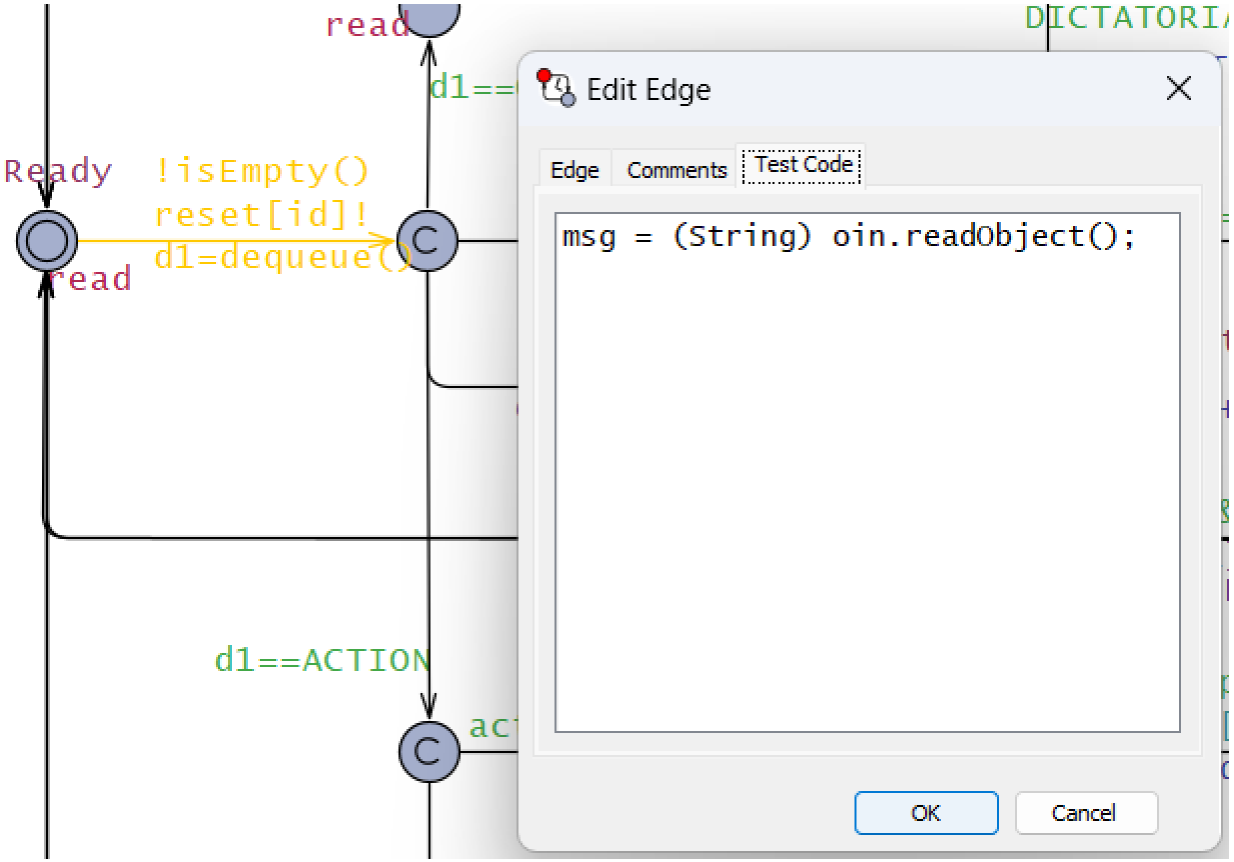
Adequacy: Traceability



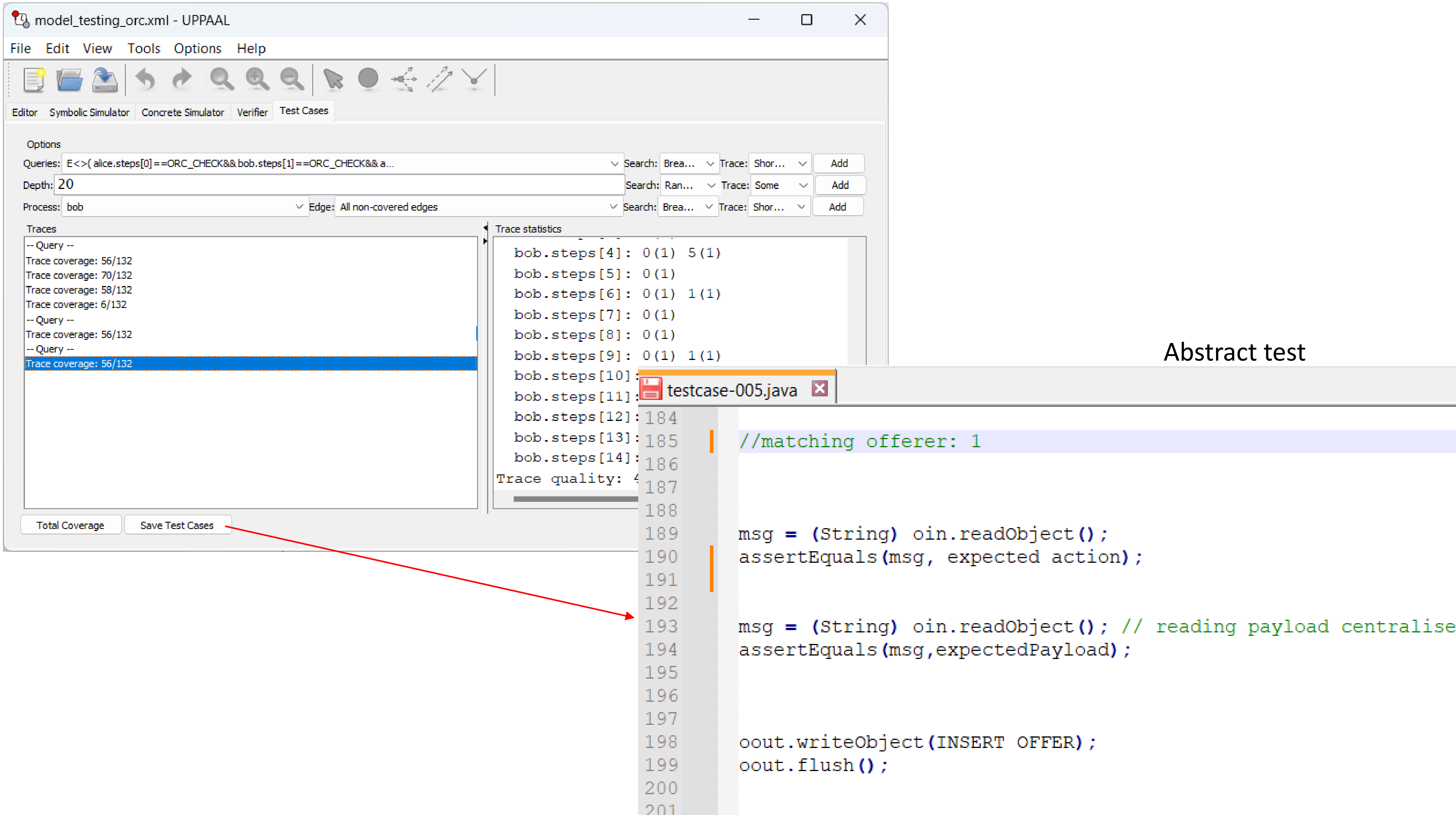
Adequacy: Traceability



Adequacy: Testing

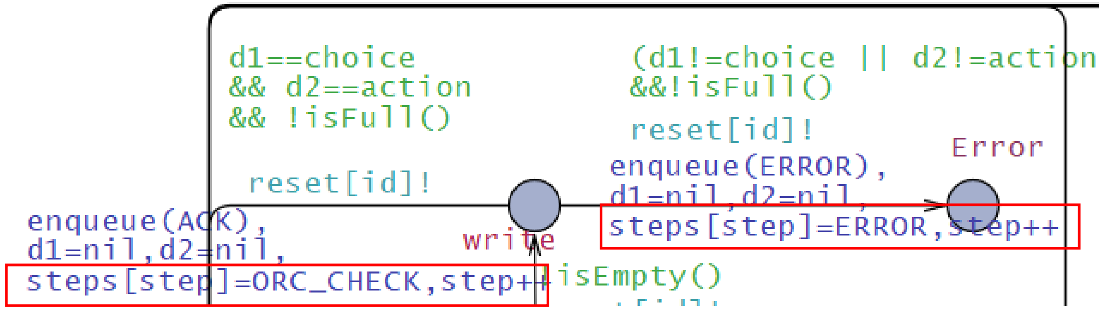


- each transition that involves enqueuing or dequeuing messages produces test code for writing to or reading from a socket, respectively,
- when running a simulation, whenever a transition is fired, the corresponding test code is appended to the abstract test case being generated.


model_testing_orc.xml - UPPAAL
File Edit View Tools Options Help
Editor
Symbolic Simulator
Concrete Simulator
Verifier
Test Cases
Options
Queries: E<>(alice.steps[0]==ORC_CHECK&& bob.steps[1]==ORC_CHECK&& a...
Search: Brea...
Trace: Shor...
Add
Depth: 20
Search: Ran...
Trace: Some
Add
Process: bob
Edge: All non-covered edges
Search: Brea...
Trace: Shor...
Add
Traces
-- Query --
Trace coverage: 56/132
Trace coverage: 70/132
Trace coverage: 58/132
Trace coverage: 6/132
-- Query --
Trace coverage: 56/132
-- Query --
Trace coverage: 56/132
Trace statistics
bob.steps[4]: 0(1) 5(1)
bob.steps[5]: 0(1)
bob.steps[6]: 0(1) 1(1)
bob.steps[7]: 0(1)
bob.steps[8]: 0(1)
bob.steps[9]: 0(1) 1(1)
bob.steps[10]:
bob.steps[11]:
bob.steps[12]:
bob.steps[13]:
bob.steps[14]:
Trace quality:
Total Coverage
Save Test Cases
Abstract test
testcase-005.java
//matching offerer: 1
msg = (String) oin.readObject();
assertEquals(msg, expected action);
msg = (String) oin.readObject(); // reading payload centralise
assertEquals(msg,expectedPayload);
oout.writeObject(INSERT OFFER);
oout.flush();

Adequacy: Testing

- Test generation from queries of the form E<>
    - encode specific simulation traces that are relevant to the specific orchestration employed in the tests
    - Additional variables utilized to encode the desired simulation in the query



Query

```
E<>(
alice.steps[0]==ORC_CHECK&&
bob.steps[1]==ORC_CHECK&&
alice.steps[2]==CENTRALISED_OFFER&&
alice.steps[3]==CENTRALISED_MATCH&&
bob.steps[4]==CENTRALISED_OFFER&&
alice.steps[5]==DICTATORIAL_CHOICE&&
bob.steps[6]==DICTATORIAL_CHOICE&&
alice.steps[7]==CENTRALISED_OFFER&&
alice.steps[8]==DICTATORIAL_CHOICE&&
bob.steps[9]==DICTATORIAL_CHOICE&&
alice.steps[10]==ORC_STOP&&
bob.steps[11]==ORC_STOP)
```

Comment

```
testing the dictatorial centralised orchestration
AlicexBob
the executed trace is (!euro,-)(?coffee,!coffee)[(!euro,-)^*]
steps 7-9 may repeat 0 or more time at runtime, the trace needs to be adjusted for this

concrete test file :  DictatorialCentralisedRunnableOrchestrationTest.java
```

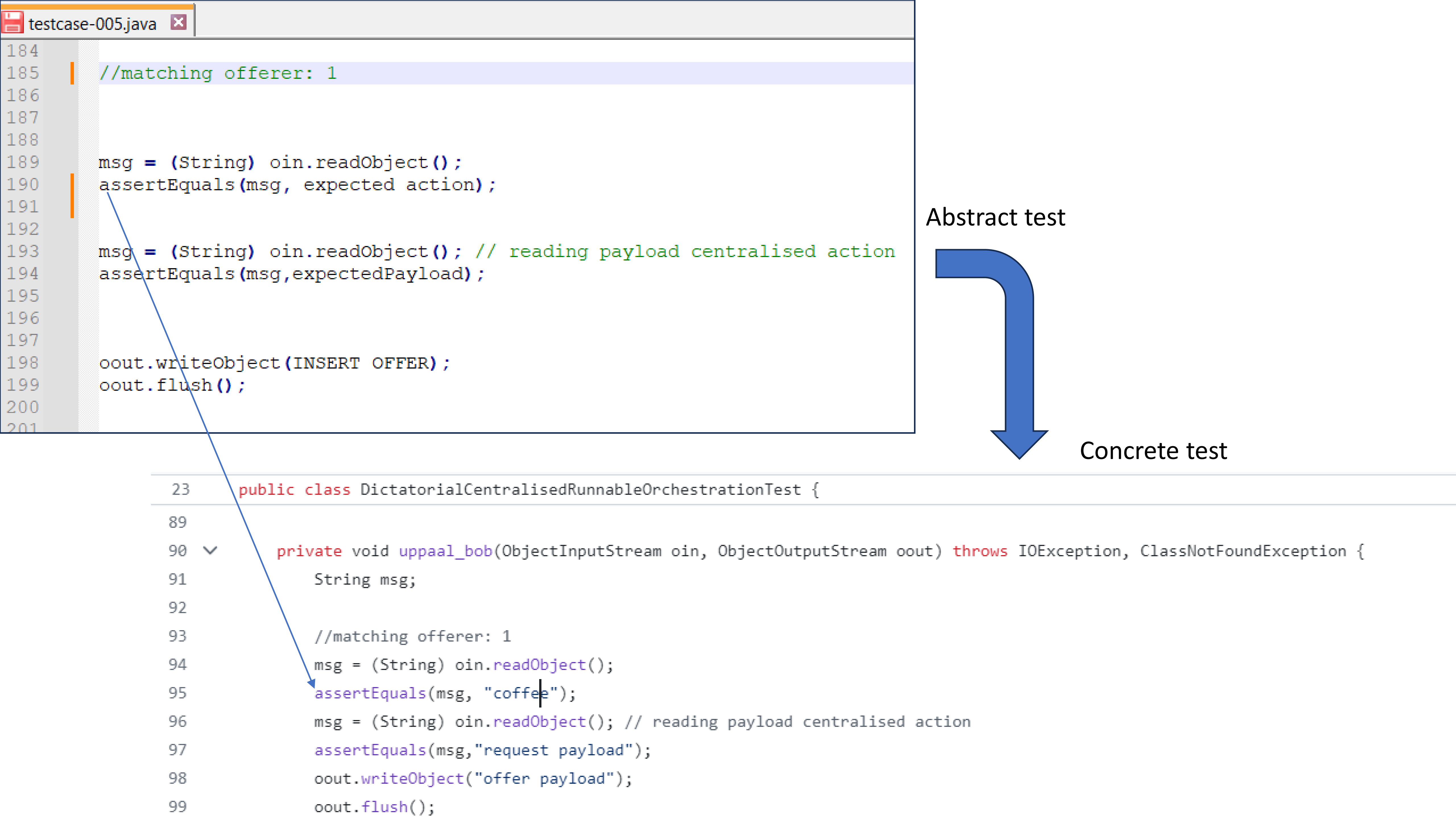
Models

- model_testing_orc.xml
- model_testing_roc.xml
- model_testing_roc_distributed_offerer.xml
- model_testing_roc_distributed_requester.xml

Concrete tests

- test
  - java/io/github/contractautomata/care/runnableOrchestration
    - Alice.java
    - Bob.java
    - DictatorialCentralisedRunnableOrchestratedContractTest.java
    - DictatorialCentralisedRunnableOrchestrationTest.java
    - DictatorialDistributedRunnableOrchestratedContractTest.java
    - DictatorialDistributedRunnableOrchestratedContract_DistributedMatchOfferer_Test.java
    - DictatorialDistributedRunnableOrchestratedContract_DistributedMatchRequester_Test.java
    - MajoritarianCentralisedRunnableOrchestratedContractTest.java
    - MajoritarianCentralisedRunnableOrchestrationTest.java
    - MajoritarianDistributedRunnableOrchestrationTest.java
  - resources
    - Alice.data
    - Alice2.data
    - Bob.data
    - Bob2.data
    - Orchestration.data
    - Orchestration2.data

- *model_testing_orc.xml* used for testing the RunnableOrchestration
  - the runnable contracts are the testers
- *model_testing_roc.xml* used for testing the RunnableOrchestratedContract
  - the tester is only the orchestrator

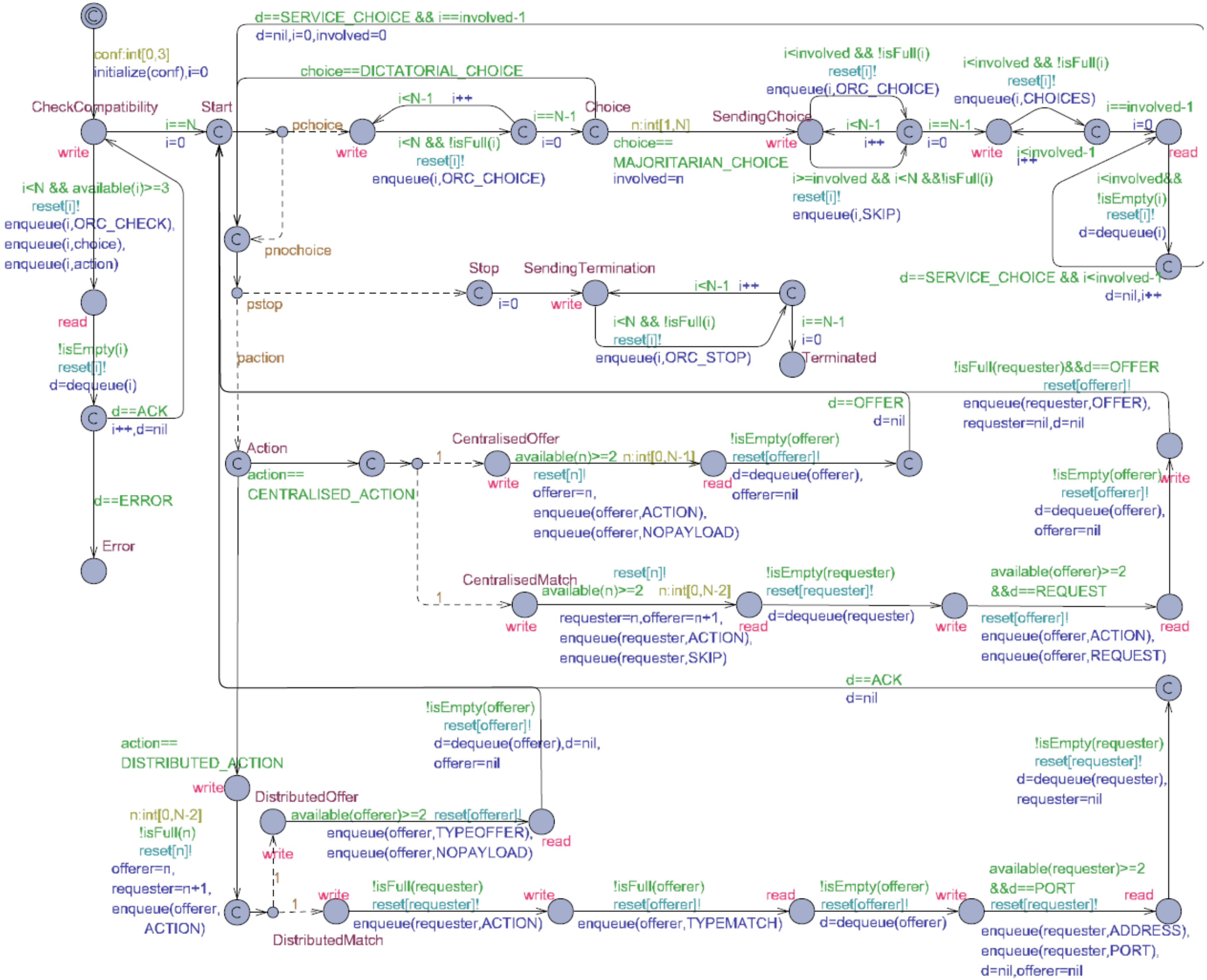

testcase-005.java
//matching offerer: 1
msg = (String) oin.readObject();
assertEquals(msg, expected action);
msg = (String) oin.readObject(); // reading payload centralised action
assertEquals(msg,expectedPayload);
oout.writeObject(INSERT OFFER);
oout.flush();
Abstract test
Concrete test
public class DictatorialCentralisedRunnableOrchestrationTest {
private void uppaal_bob(ObjectInputStream oin, ObjectOutputStream oout) throws IOException, ClassNotFoundException {
String msg;
//matching offerer: 1
msg = (String) oin.readObject();
assertEquals(msg, "coffee");
msg = (String) oin.readObject(); // reading payload centralised action
assertEquals(msg,"request payload");
oout.writeObject("offer payload");
oout.flush();

Adequacy: Testing

- the generated tests cover all transitions of the model and all interactions between the orchestrator and the services
- the code coverage indicates that the tests derived from the model cover a significant portion of the source code
- the model is not excessively abstract compared to the actual implementation.

coverage

Element	Class, %	Method, %	Line, %
io.github.contractautomata.care.runnable	100% (12/12)	84% (39/46)	83% (292/351)
actions	100% (4/4)	100% (8/8)	92% (91/98)
CentralisedOrchestratedAction	100% (1/1)	100% (2/2)	100% (8/8)
CentralisedOrchestratorAction	100% (1/1)	100% (2/2)	95% (22/23)
DistributedOrchestratedAction	100% (1/1)	100% (2/2)	88% (39/44)
DistributedOrchestratorAction	100% (1/1)	100% (2/2)	95% (22/23)
OrchestratedAction	100% (0/0)	100% (0/0)	100% (0/0)
OrchestratorAction	100% (0/0)	100% (0/0)	100% (0/0)
choice	100% (4/4)	81% (13/16)	90% (57/63)
DictatorialChoiceRunnableOrchest	100% (1/1)	100% (3/3)	100% (3/3)
DictatorialChoiceRunnableOrchest	100% (1/1)	75% (3/4)	83% (10/12)
MajoritarianChoiceRunnableOrche	100% (1/1)	100% (4/4)	94% (16/17)
MajoritarianChoiceRunnableOrche	100% (1/1)	60% (3/5)	90% (28/31)
AutoCloseableList	100% (1/1)	100% (2/2)	71% (5/7)
RunnableOrchestratedContract	100% (2/2)	80% (8/10)	80% (69/86)
RunnableOrchestration	100% (1/1)	80% (8/10)	72% (70/97)



d==SERVICE_CHOICE && i==involved-1
d=nil,i=0,involved=0
conf:int[0,3]
initialize(conf),i=0
choice==DICTATORIAL_CHOICE
CheckCompatibility
Start
i==N
i=0
write
pchoice
i<N-1 i++
i==N-1
i=0
Choice
i<N && !isFull(i)
reset[i]!
enqueue(i,ORC_CHOICE)
n:int[1,N]
choice==
MAJORITARIAN_CHOICE
involved=n
SendingChoice
i<involved && !isFull(i)
reset[i]!
enqueue(i,ORC_CHOICE)
i<N-1
i++
i==N-1
i=0
i>=involved && i<N &&!isFull(i)
reset[i]!
enqueue(i,SKIP)
i<involved && !isFull(i)
reset[i]!
enqueue(i,CHOICES)
i==involved-1
i=0
i<involved-1
i++
read
i<involved&&
!isEmpty(i)
reset[i]!
d=dequeue(i)
d==SERVICE_CHOICE && i<involved-1
d=nil,i++
i<N && available(i)>=3
reset[i]!
enqueue(i,ORC_CHECK),
enqueue(i,choice),
enqueue(i,action)
pnochoice
Stop
SendingTermination
i<N-1 i++
pstop
i=0
i<N && !isFull(i)
reset[i]!
enqueue(i,ORC_STOP)
i==N-1
i=0
Terminated
!isEmpty(i)
reset[i]!
d=dequeue(i)
paction
d==ACK
i++,d=nil
d==ERROR
Error
!isFull(requester)&&d==OFFER
reset[offerer]!
enqueue(requester,OFFER),
requester=nil,d=nil
d==OFFER
d=nil
Action
action==
CENTRALISED_ACTION
CentralisedOffer
1
available(n)>=2 n:int[0,N-1]
reset[n]!
offerer=n,
enqueue(offerer,ACTION),
enqueue(offerer,NOPAYLOAD)
!isEmpty(offerer)
reset[offerer]!
d=dequeue(offerer),
offerer=nil
!isEmpty(offerer)
reset[offerer]!
d=dequeue(offerer),
offerer=nil
CentralisedMatch
reset[n]!
available(n)>=2 n:int[0,N-2]
requester=n,offerer=n+1,
enqueue(requester,ACTION),
enqueue(requester,SKIP)
!isEmpty(requester)
reset[requester]!
d=dequeue(requester)
available(offerer)>=2
&&d==REQUEST
reset[offerer]!
enqueue(offerer,ACTION),
enqueue(offerer,REQUEST)
d==ACK
d=nil
!isEmpty(offerer)
reset[offerer]!
d=dequeue(offerer),d=nil,
offerer=nil
action==
DISTRIBUTED_ACTION
DistributedOffer
available(offerer)>=2 reset[offerer]!
enqueue(offerer,TYPEOFFER),
enqueue(offerer,NOPAYLOAD)
n:int[0,N-2]
!isFull(n)
reset[n]!
offerer=n,
requester=n+1,
enqueue(offerer,
ACTION)
!isFull(requester)
reset[requester]!
enqueue(requester,ACTION)
!isFull(offerer)
reset[offerer]!
enqueue(offerer,TYPEMATCH)
!isEmpty(offerer)
reset[offerer]!
d=dequeue(offerer)
available(requester)>=2
&&d==PORT
reset[requester]!
enqueue(requester,ADDRESS),
enqueue(requester,PORT),
d=nil,offerer=nil
!isEmpty(requester)
reset[requester]!
d=dequeue(requester),
requester=nil
DistributedMatch

Analysis: modelling phase

- Validation through modelling:
  - an undetected issue in the source code was identified during the modelling phase, related to the majoritarian choice
  - The orchestrator was waiting for a message also from the services not involved in the choice
  - The issue was undetected because in all tests all services were involved in a choice

Analysis: parameters tuning

- Delays in reading and writing, timeout, buffer size, probability weights
- Goals:
  - realistic modelling: low probability of filling the buffers, timeout, excessive delays
  - improved verification performances: reducing the state-space of the model for the exhaustive verification

- Probability weights (e.g., pchoice, paction,..) can be fine tuned to model an orchestration or a set of orchestrations

Parameters tuning: buffer size

```
E[<=500; 10000](max: sum(i:int[0,N-1]) (sum(j:int[0,queueSize-1])(orc2services[i][j]!=nil)))
```

- Goal: prevent unnecessary growth in the state space whilst reducing the probability of filling the buffers
  - Default size of Java TCP/IP Sockets is 8 KB
- With buffer size=10, the formula evaluates to ~ 4.5
  - The buffer size can be safely reduced in the model

```
Pr[<=500] (<>(exists (i:id_t) ror.isFull(i)))
```

- Evaluates to [0,0.00996915] with buffer size set to 5
  - The buffer size is set to 5 for the subsequent experiments

Parameters tuning: timeout

- Selected configuration of rates and timeout
- Goals: low probability of timeout, high probability of terminating within a given timeframe, lower timeout threshold

```
Pr[<=500] (<> ror.Timeout)
```

```
Pr[<=500](<>ror.Terminated&&(forall (i:id_t) ROC(i).Terminated))
```

- Exhaustive model checking: (#services, buffer size) either set to (4,5) or (5,3). The configuration (5,4) remained inconclusive.

Formal verification

- Termination

```
ror.Stop-->((ror.Terminated&&(forall(i:id_t)ROC(i).Terminated))
              ||(exists(i:id_t)SocketTimeout(i).Timeout))
```

  - P - - > Q is a shortcut for A[](p imply A<>q)

- Absence of deadlocks

```
A[](not deadlock || (exists(i:id_t) SocketTimeout(i).Timeout) ||
     (ror.Terminated && (forall (i:id_t) ROC(i).Terminated)))
```

  - no error state is ever reached
  - an error in the model has been detected and fixed by model checking this formula

Formal verification

- Absence of orphan messages

```
Pr[<=500](<>!allEmpty()&&ror.Terminated&&(forall(i:id_t)ROC(i).Terminated))
```

```
A[]((ror.Terminated && (forall (i:id_t) ROC(i).Terminated)) imply allEmpty())
```

- No dummy execution

```
E[] (allEmpty() && !ror.Timeout)
```

Formal Verification

- No interference in matches

```
Pr[<=500](<>exists(i:id_t)(i<N-1&&(ROC(i).d1==TYPEMATCH||ROC(i).d1==ADDRESS||ROC(i).d1==PORT))
           &&((ROC(i+1).d1==TYPEMATCH|| ROC(i+1).d1==ADDRESS||ROC(i+1).d1==PORT))&&
(exists(j:id_t)(j!=i&&j!=i+1&&(ROC(j).d1==TYPEMATCH||ROC(j).d1==ADDRESS||ROC(j).d1==PORT)))
```

Formal Verification

- Mismatching configurations

```
ror = RunnableOrchestration(MAJORITARIAN_CHOICE,DISTRIBUTED_ACTION);
alice = RunnableOrchestratedContract(0,MAJORITARIAN_CHOICE,DISTRIBUTED_ACTION);
bob = RunnableOrchestratedContract(1,MAJORITARIAN_CHOICE,DISTRIBUTED_ACTION);
carl = RunnableOrchestratedContract(2,DICTATORIAL_CHOICE,DISTRIBUTED_ACTION);
ast = SocketTimeout(0);bst = SocketTimeout(1);cst = SocketTimeout(2);
system ror,alice,bob,carl,ast,bst,cst;
```

```
A<>((ror.Error && carl.Error)||ror.Timeout)
```

```
A[](!ror.Start)
```

Conclusion

- Modelling, Verification and Testing of CARE
  - model-based testing and traceability for validating the adequacy of the model
  - Statistical model checking and exhaustive model checking for fine-tuning the parameters and perform the verification

- Future work:
  - automatic alignment of artifacts,
  - managing configurations (UPPEX).

Finanziato dall'Unione europea
NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

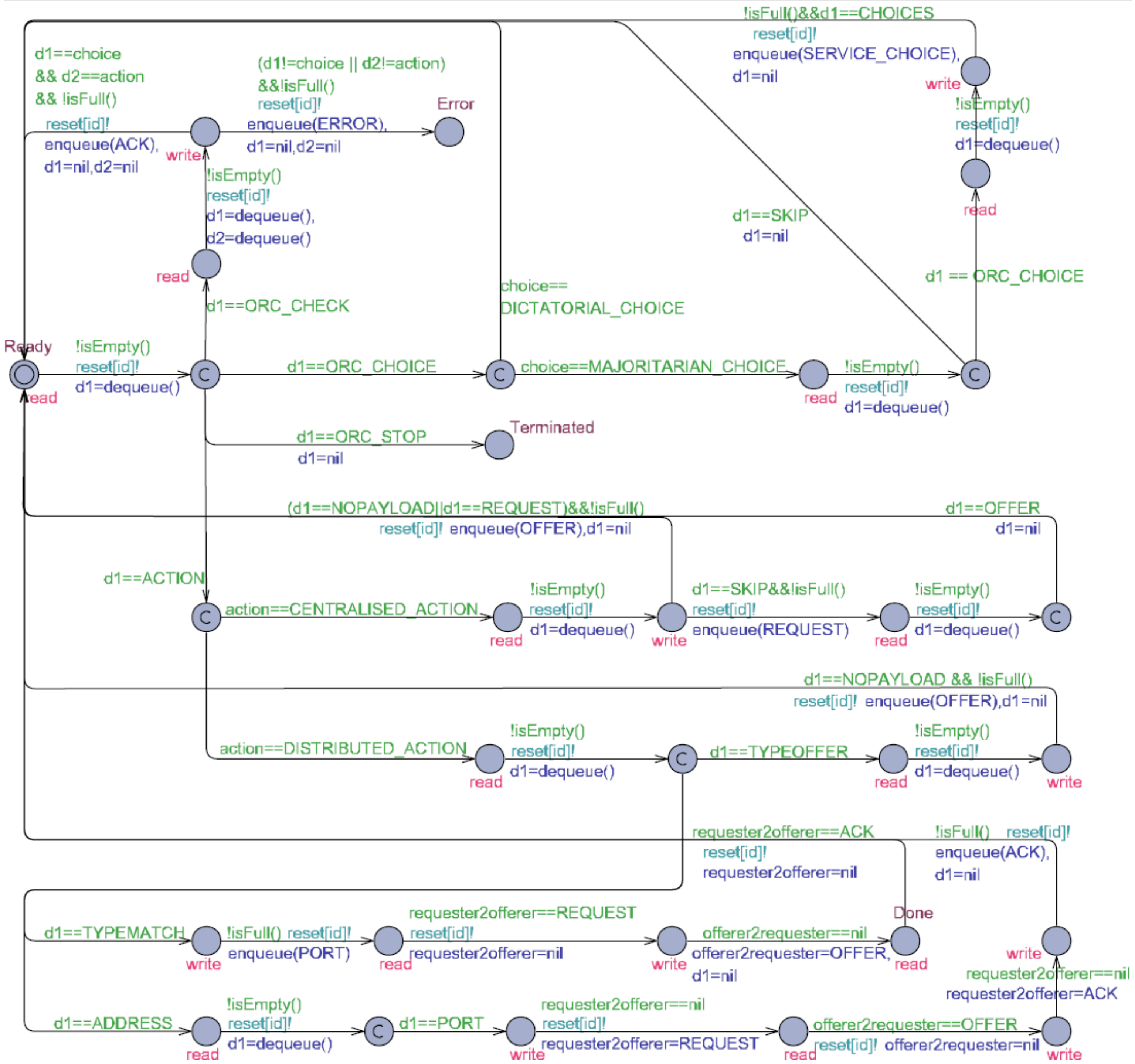
Conclusion

- Modelling, Verification and Testing of CARE
  - model-based testing and traceability for validating the adequacy of the model
  - Statistical model checking and exhaustive model checking for fine-tuning the parameters and perform the verification

- Future work:
  - automatic alignment of artifacts,
  - managing configurations (UPPEX).

- Thanks for your attention

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA



!isFull()&&d1==CHOICES
reset[id]!
enqueue(SERVICE_CHOICE),
d1=nil
write
d1==choice
&& d2==action
&& !isFull()
reset[id]!
enqueue(ACK),
d1=nil,d2=nil
write
(d1!=choice || d2!=action)
&&!isFull()
reset[id]!
enqueue(ERROR),
d1=nil,d2=nil
Error
!isEmpty()
reset[id]!
d1=dequeue()
!isEmpty()
reset[id]!
d1=dequeue(),
d2=dequeue()
read
read
d1==SKIP
d1=nil
d1 == ORC_CHOICE
choice==
DICTATORIAL_CHOICE
d1==ORC_CHECK
Ready
!isEmpty()
reset[id]!
d1=dequeue()
read
d1==ORC_CHOICE
choice==MAJORITARIAN_CHOICE
read
!isEmpty()
reset[id]!
d1=dequeue()
Terminated
d1==ORC_STOP
d1=nil
(d1==NOPAYLOAD||d1==REQUEST)&&!isFull()
reset[id]! enqueue(OFFER),d1=nil
d1==OFFER
d1=nil
d1==ACTION
action==CENTRALISED_ACTION
read
!isEmpty()
reset[id]!
d1=dequeue()
write
d1==SKIP&&!isFull()
reset[id]!
enqueue(REQUEST)
read
!isEmpty()
reset[id]!
d1=dequeue()
d1==NOPAYLOAD && !isFull()
reset[id]! enqueue(OFFER),d1=nil
action==DISTRIBUTED_ACTION
read
!isEmpty()
reset[id]!
d1=dequeue()
d1==TYPEOFFER
read
!isEmpty()
reset[id]!
d1=dequeue()
write
requester2offerer==ACK
reset[id]!
requester2offerer=nil
!isFull() reset[id]!
enqueue(ACK),
d1=nil
d1==TYPEMATCH
write
!isFull() reset[id]!
enqueue(PORT)
read
requester2offerer==REQUEST
reset[id]!
requester2offerer=nil
write
offerer2requester==nil
offerer2requester=OFFER,
d1=nil
Done
read
write
requester2offerer==nil
requester2offerer=ACK
d1==ADDRESS
read
!isEmpty()
reset[id]!
d1=dequeue()
d1==PORT
write
requester2offerer==nil
reset[id]!
requester2offerer=REQUEST
read
offerer2requester==OFFER
reset[id]! offerer2requester=nil
write