Automatic Generation of Valid Behavioral Scripts from UML Sequence Diagrams

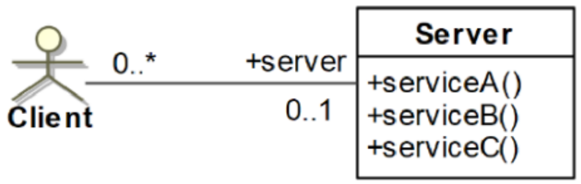
Paula Muñoz, Loli Burgueño, Antonio Vallecillo, and Martin Gogolla

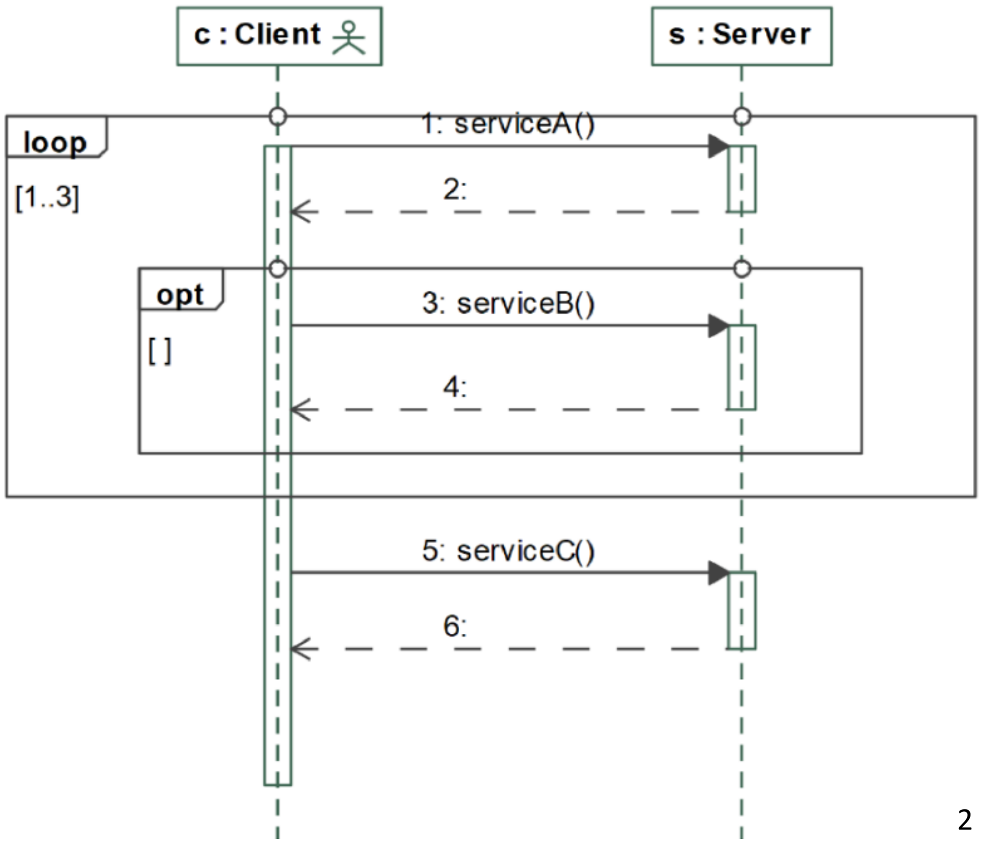
OCL and Textual Modeling Workshop @ MODELS 2019

Munich, Germany, September 16, 2019

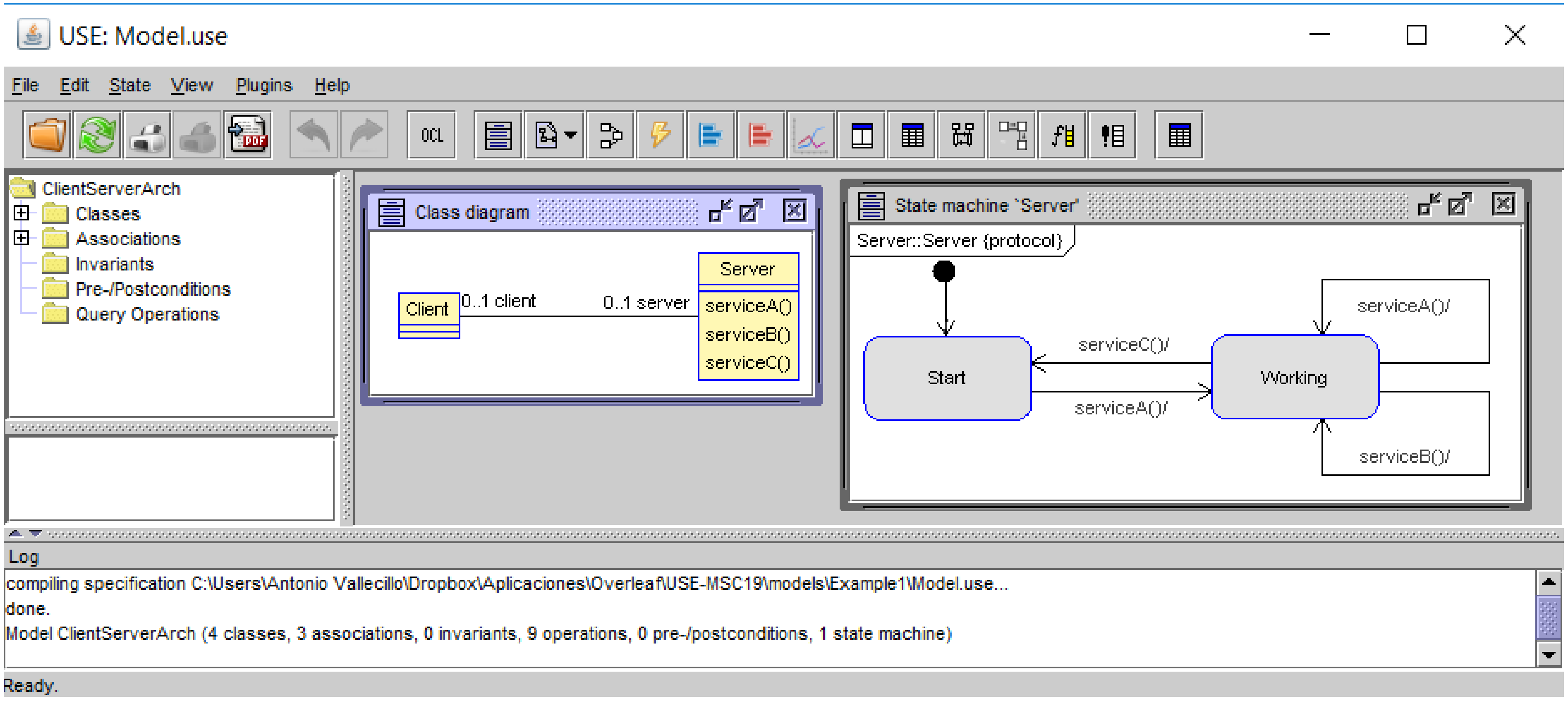
UML sequence diagrams

- Specify (valid) partial order of message interchanges between objects
- Modularity mechanisms
- Semantics defined in terms of valid and invalid traces
- Much more complex to understand than they seem!!!!





UML-based Specification Environment (USE)



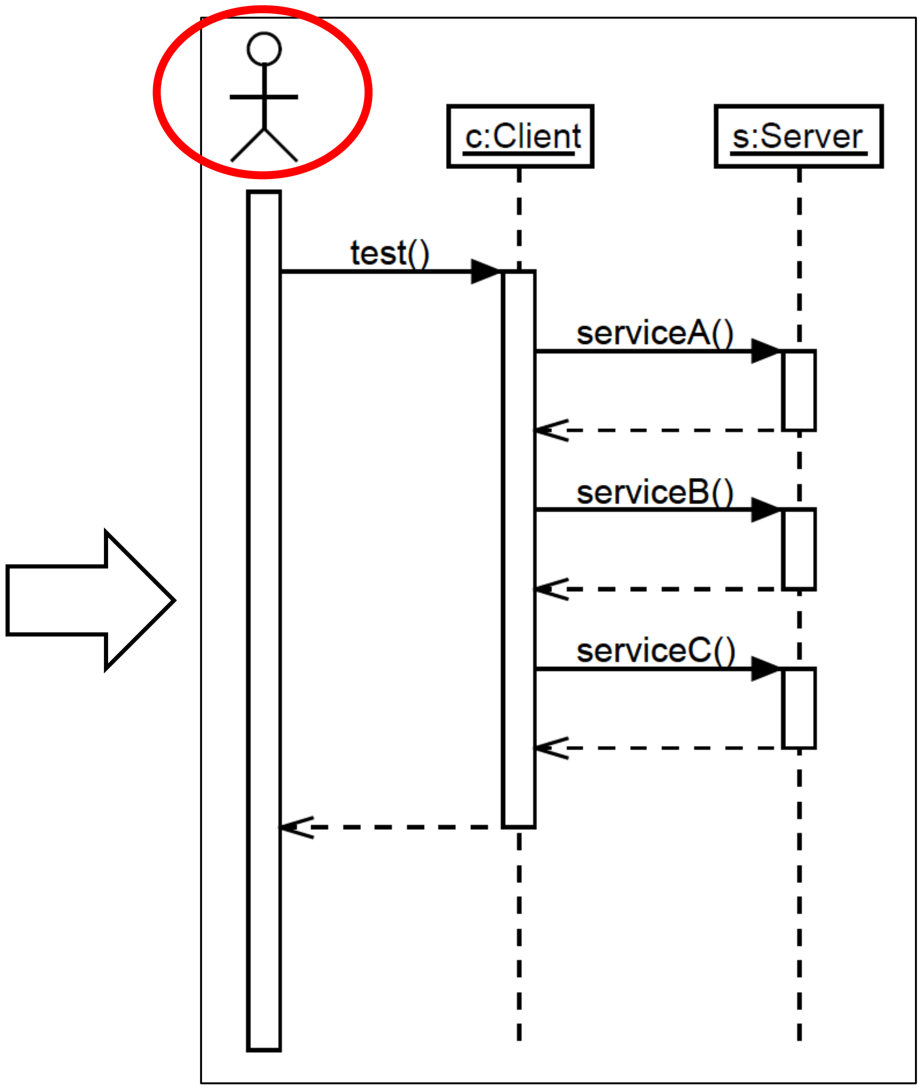
```
compiling specification C:\Users\Antonio Vallecillo\Dropbox\Aplicaciones\Overleaf\USE-MSC19\models\Example1\Model.use...
done.
Model ClientServerArch (4 classes, 3 associations, 0 invariants, 9 operations, 0 pre-/postconditions, 1 state machine)
```

Ready.

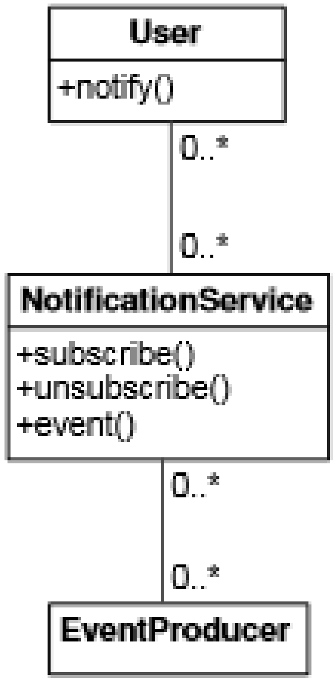
Behavior in USE

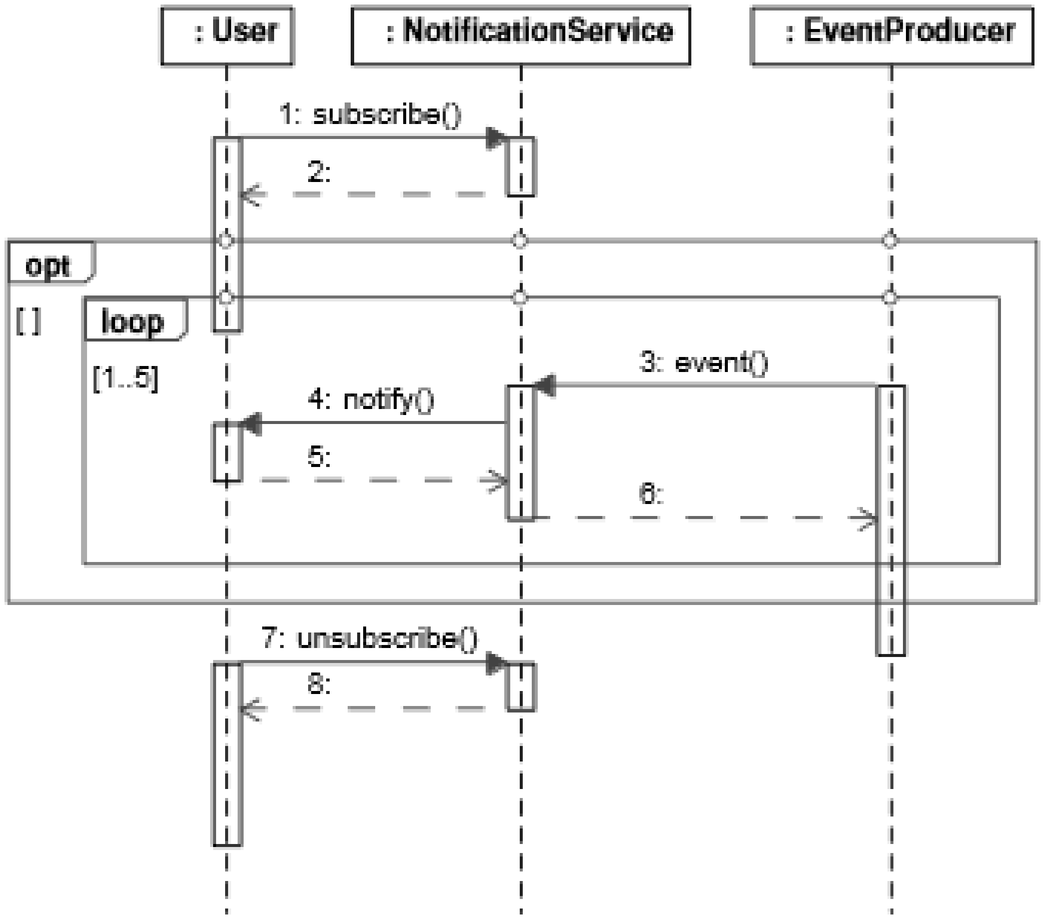
```
class Client
operations
test()
begin
  self.server.serviceA();
  self.server.serviceB();
  self.server.serviceC();
end
```

```
!new Client ('c');
!new Server ('s');
!insert (c, s) into CS;
!c.test()
```



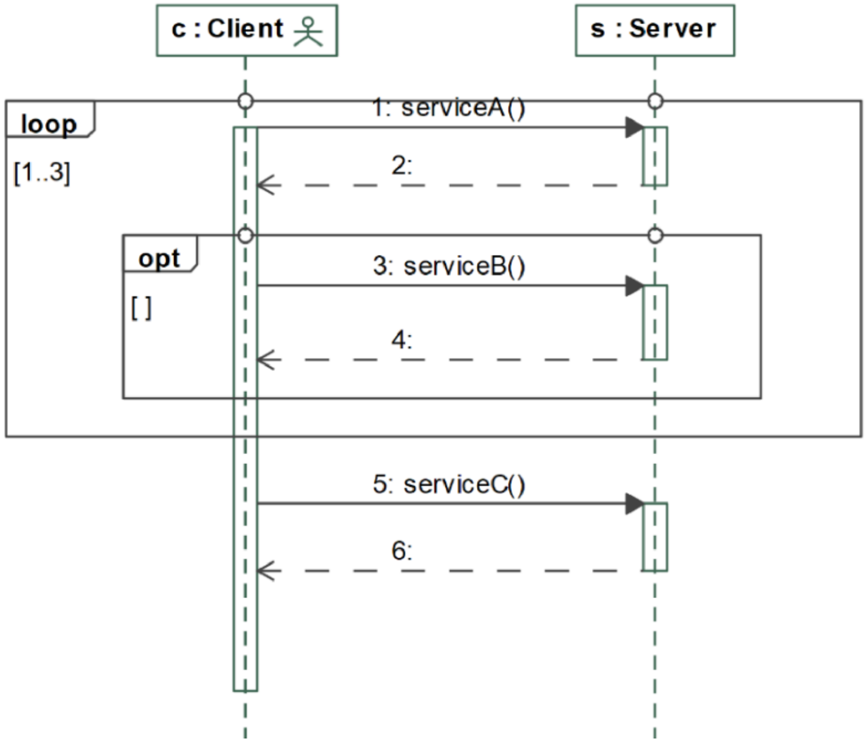
How to deal with arbitraty UML sequence diagrams?



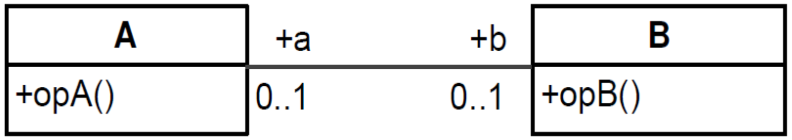


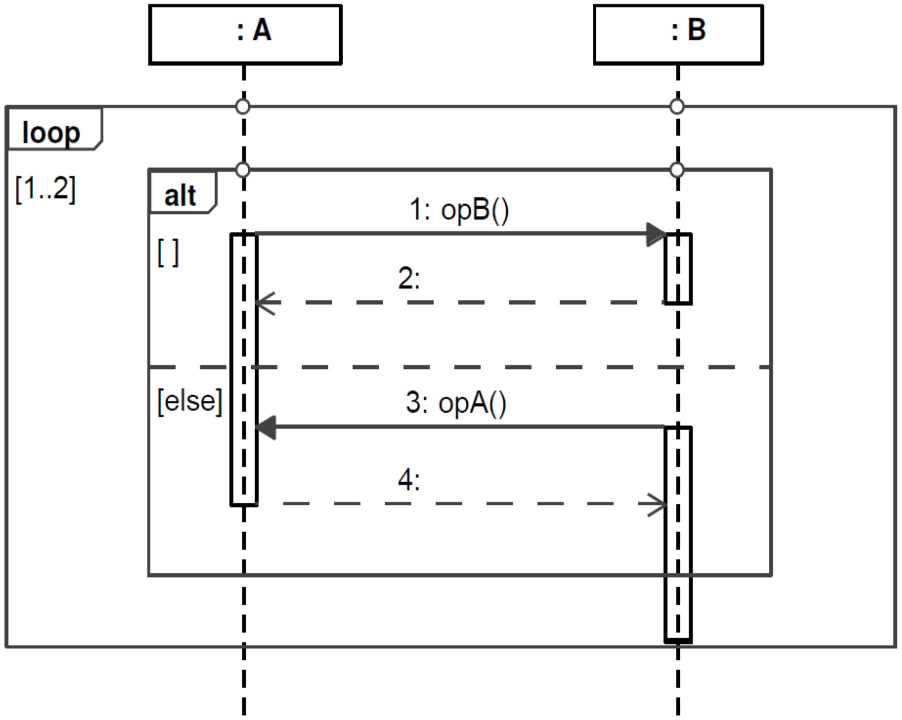
Message Sequence Charts (MSC) – ITU-T Rec. Z-120

```
msc Example1;
    (inst c:Client, s:Server)
    c,s : loop<1,3> begin loop1;
        c : call serviceA() to s;
        s : receive serviceA() from c;
        s : replyout serviceA() to c;
        c : replyin serviceA() from s;
        c,s : opt begin opt1;
            c : call serviceB() to s;
            s : receive serviceB() from c;
            s : replyout serviceB() to c;
            c : replyin serviceB() from s;
        opt end;
    loop end;
    c : call serviceC() to s;
    s : receive serviceC() from c;
    s : replyout serviceC() to c;
    c : replyin serviceC() from s;
endmsc;
```



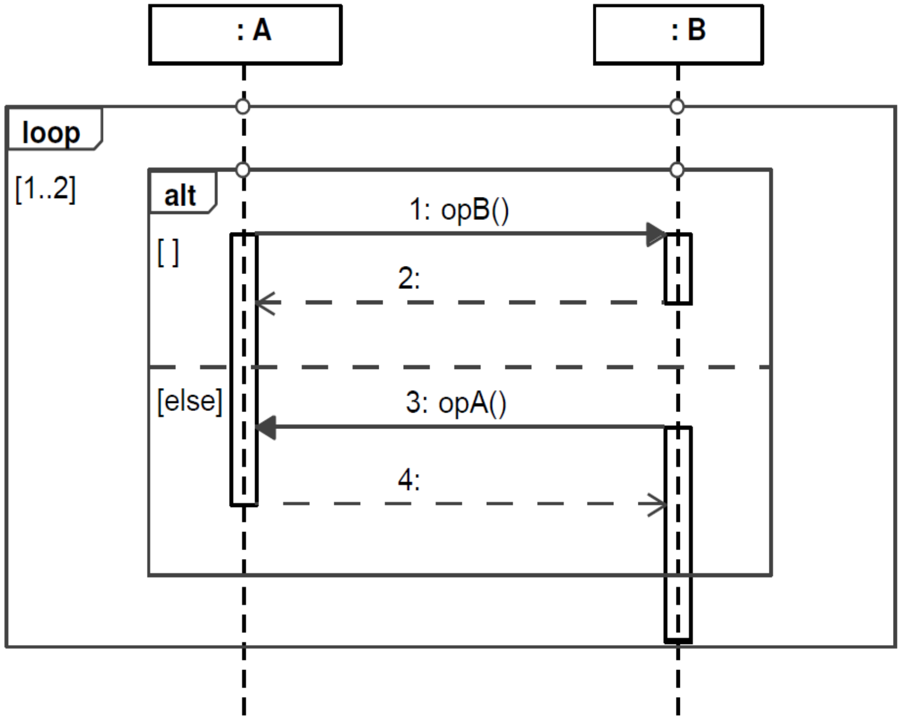
A simple example





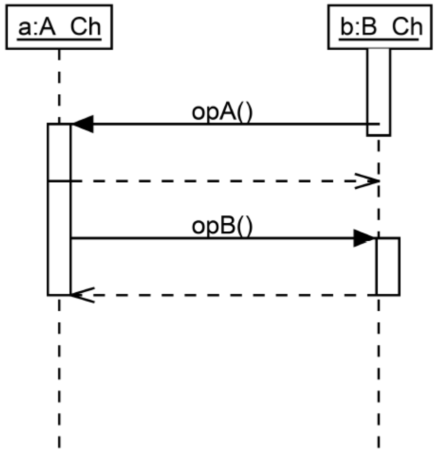
Message Sequence Charts (MSC) – ITU-T Rec. Z-120

```
msc Example2;
  (inst a:A, b:B)
  a,b : loop<1,2> begin loop1;
    a,b : alt begin alt1;
        a : call b() to b;
        b : receive b() from a;
        b : replyout b() to a;
        a : replyin b() from b;
     alt;
        b : call a() to a;
        a : receive a() from b;
        a : replyout a() to b;
        b : replyin a() from a;
     alt end;
  loop end;
endmsc;
```

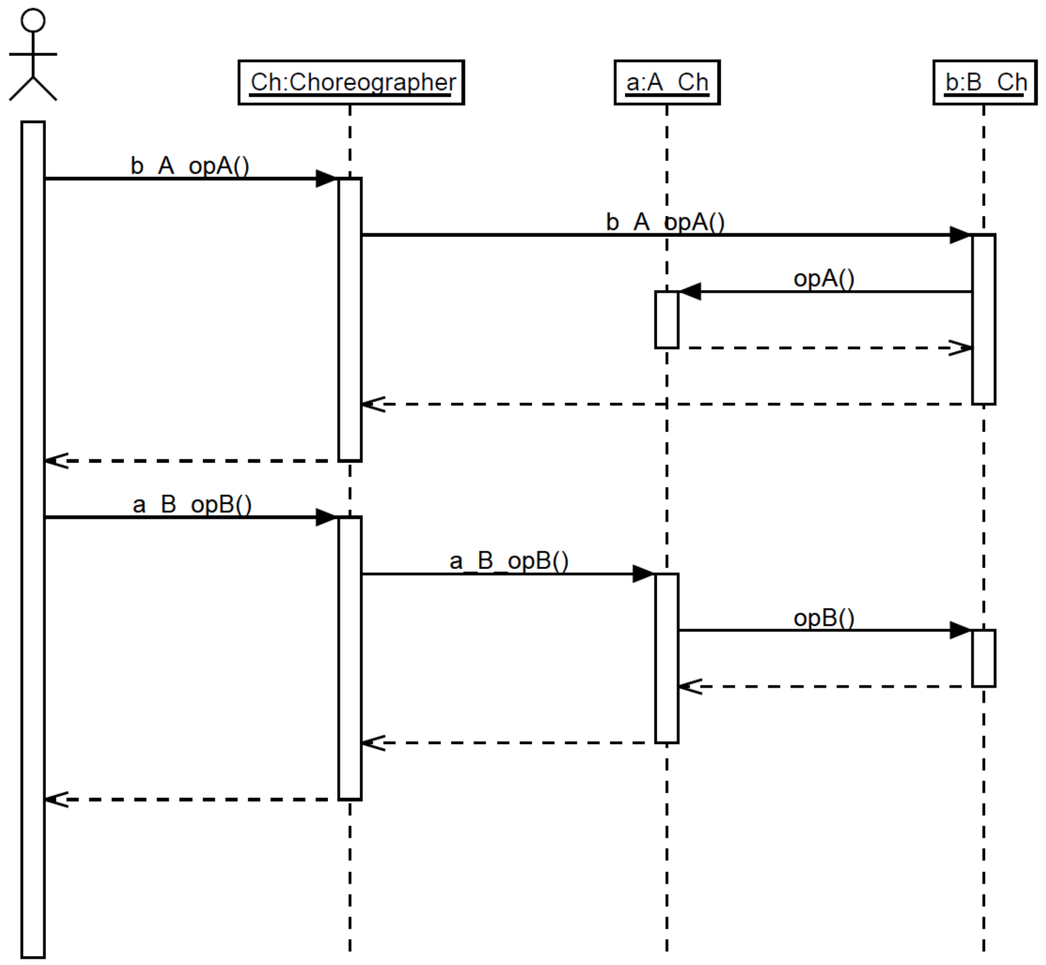


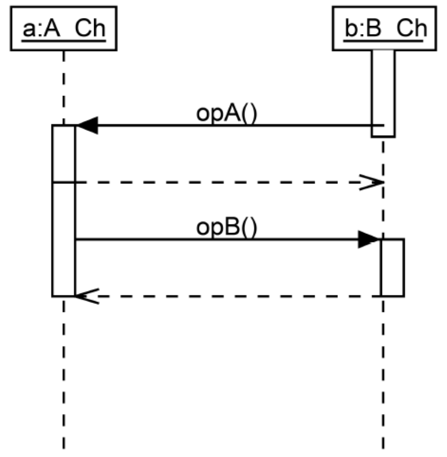
One of the resulting executions in USE

- But…
- …how can we get such a behavior in USE?

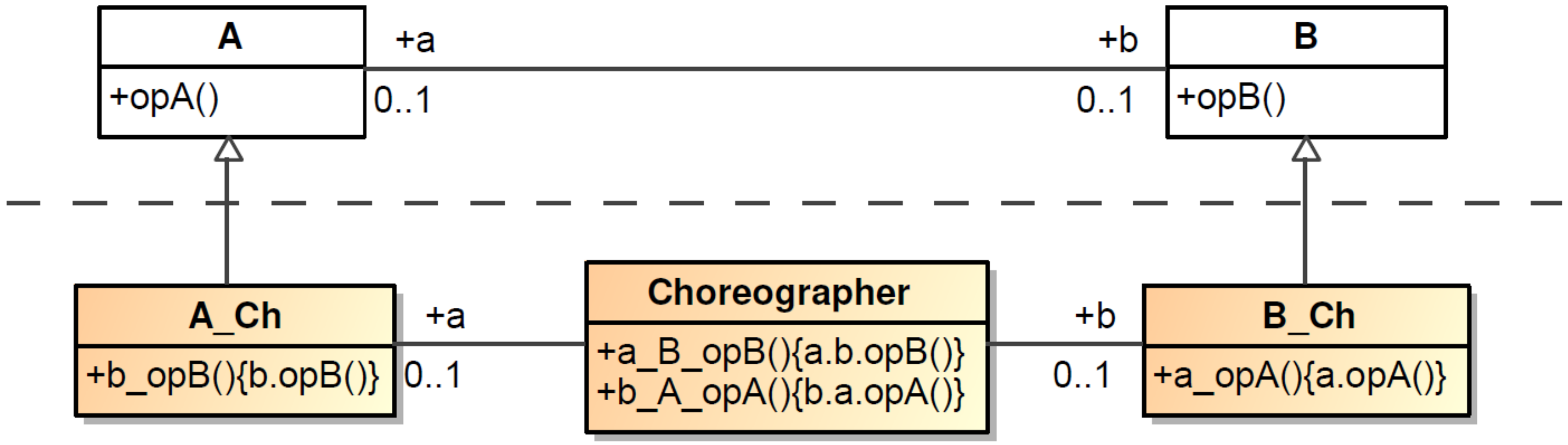


One of the resulting executions in USE (with choreography)

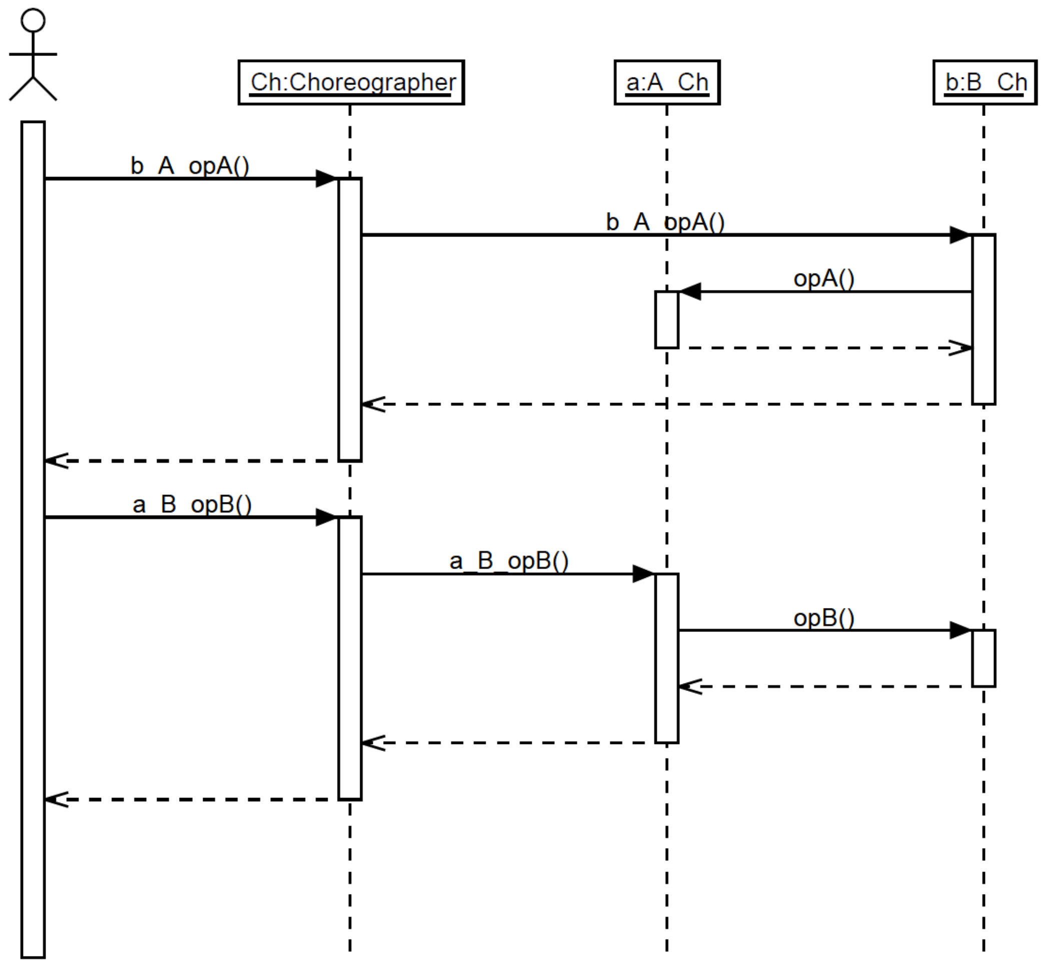


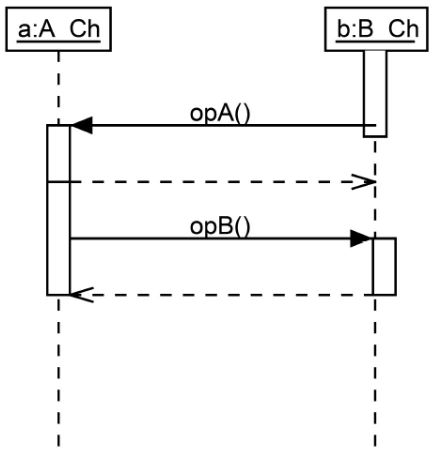


Extending the original objects to allow different choreographies

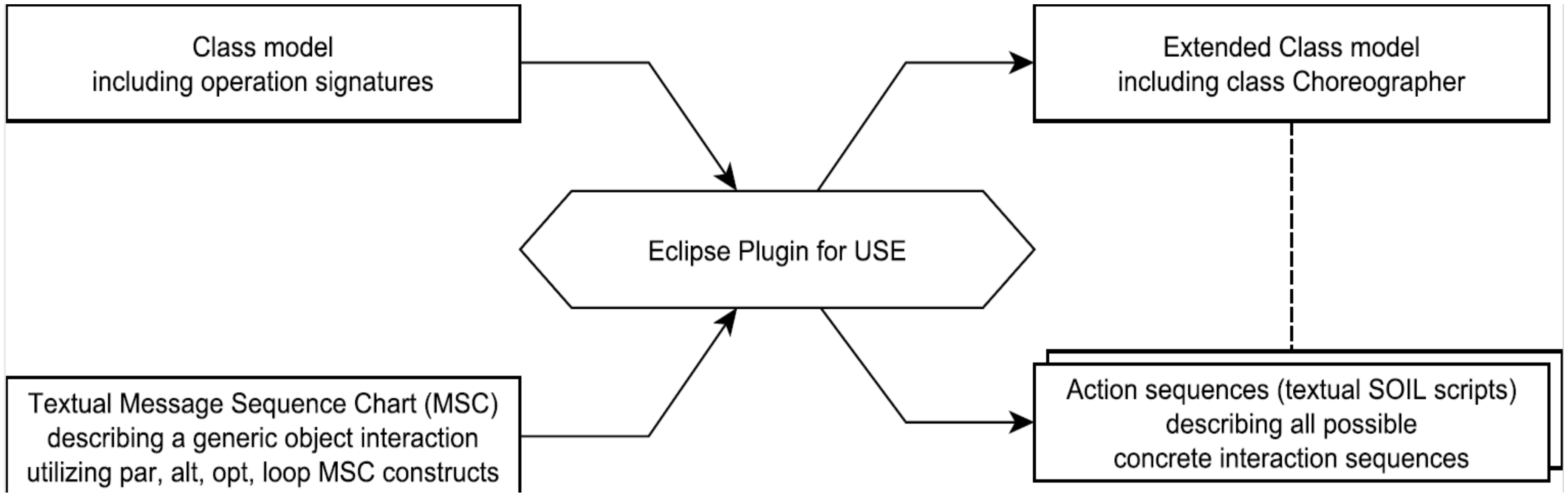


One of the resulting executions in USE

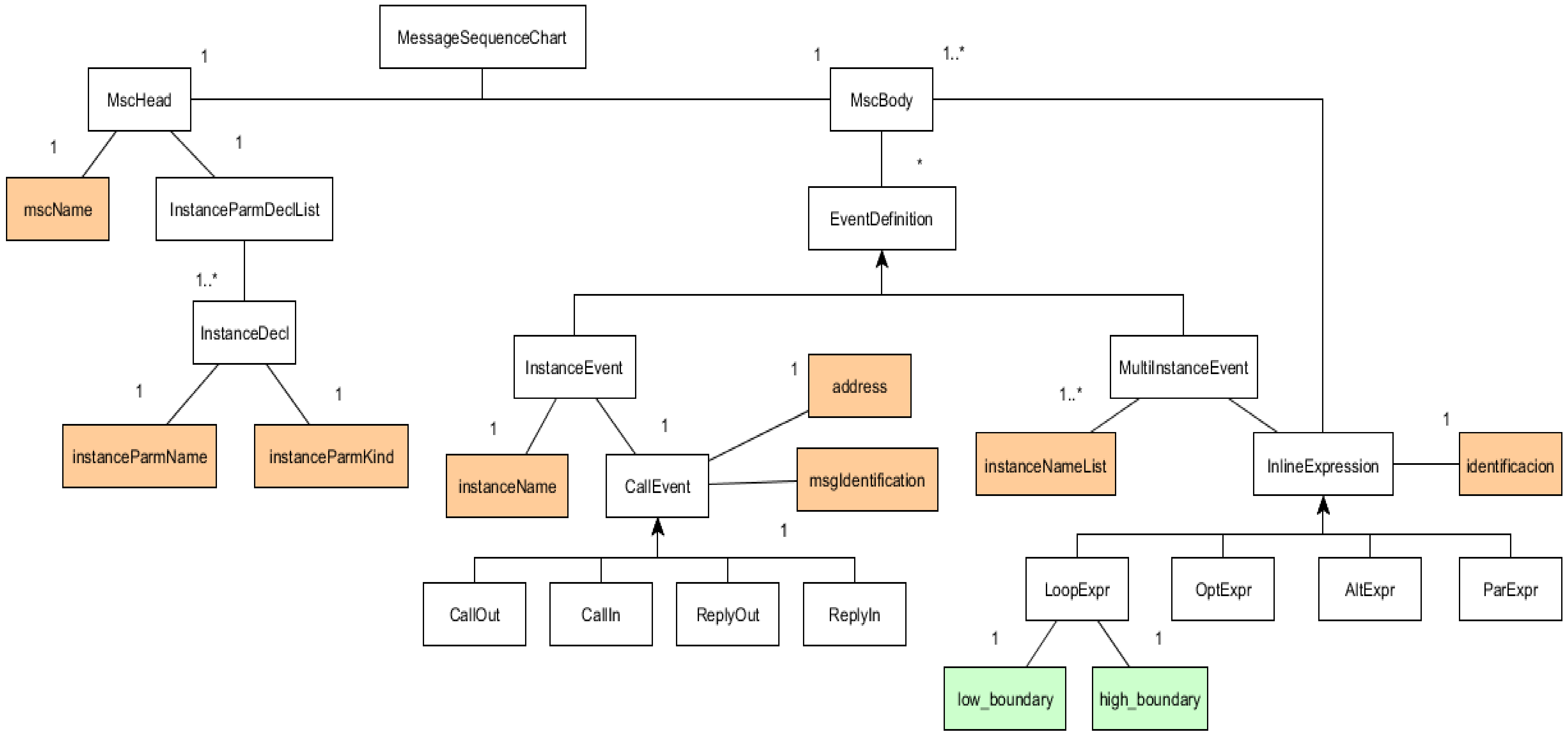




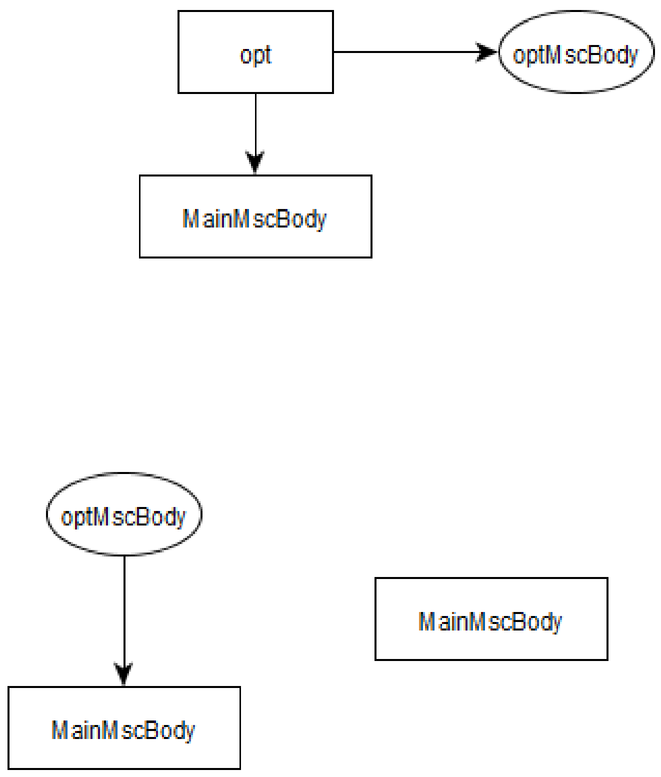
Our solution

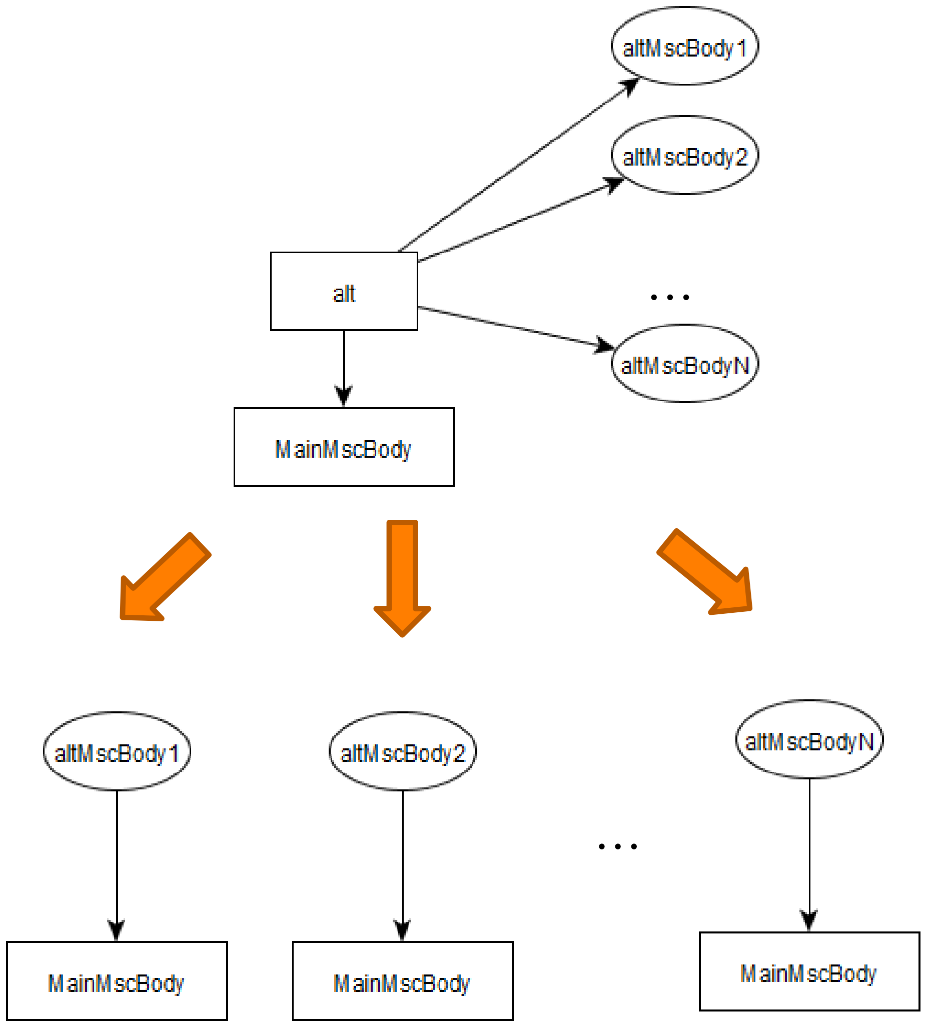

Class model
including operation signatures
Textual Message Sequence Chart (MSC)
describing a generic object interaction
utilizing par, alt, opt, loop MSC constructs
Eclipse Plugin for USE
Extended Class model
including class Choreographer
Action sequences (textual SOIL scripts)
describing all possible
concrete interaction sequences

Current AST for MSCs

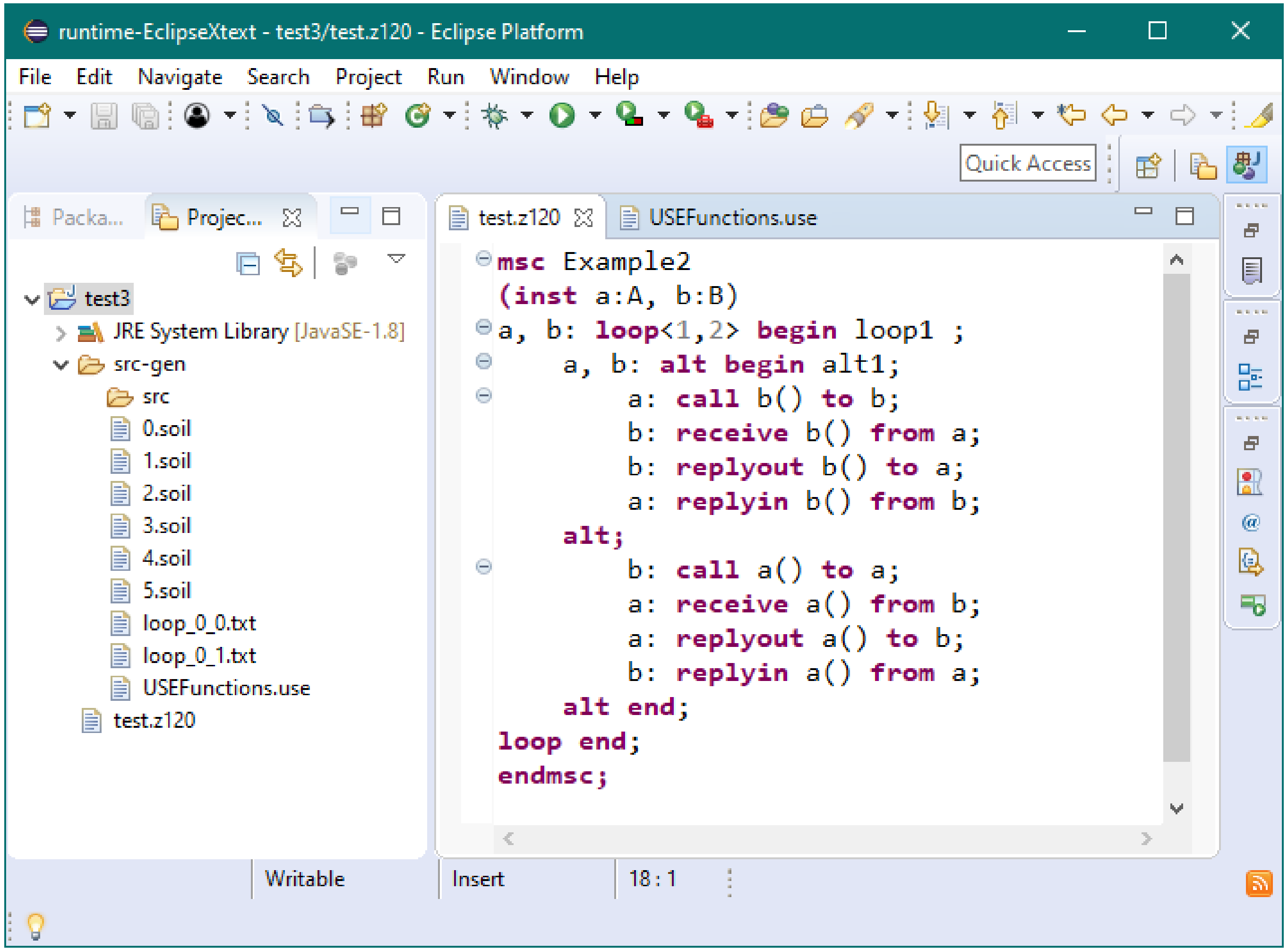


Generating SOIL scripts





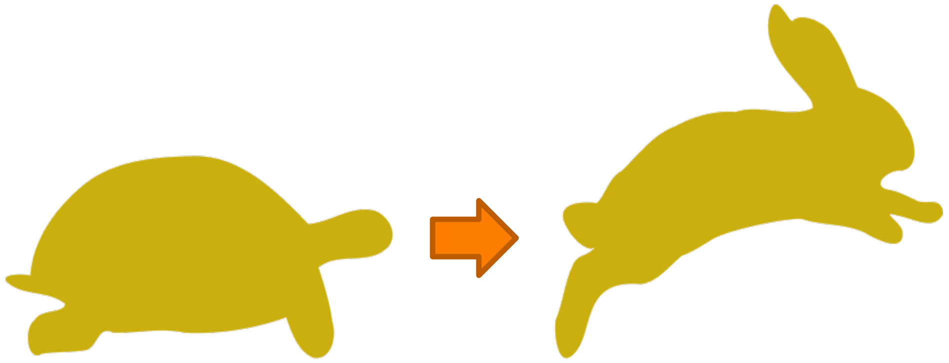
The Eclipse plugin


runtime-EclipseXtext - test3/test.z120 - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Quick Access
Packa... Projec...
test3
JRE System Library [JavaSE-1.8]
src-gen
src
0.soil
1.soil
2.soil
3.soil
4.soil
5.soil
loop_0_0.txt
loop_0_1.txt
USEFunctions.use
test.z120
test.z120 USEFunctions.use
Writable Insert 18 : 1

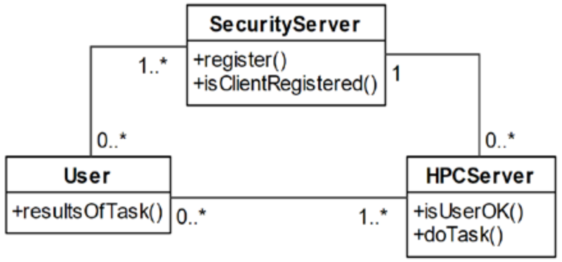
```
msc Example2
(inst a:A, b:B)
a, b: loop<1,2> begin loop1 ;
    a, b: alt begin alt1;
        a: call b() to b;
        b: receive b() from a;
        b: replyout b() to a;
        a: replyin b() from b;
    alt;
        b: call a() to a;
        a: receive a() from b;
        a: replyout a() to b;
        b: replyin a() from a;
    alt end;
loop end;
endmsc;
```

Optimization for nested loops

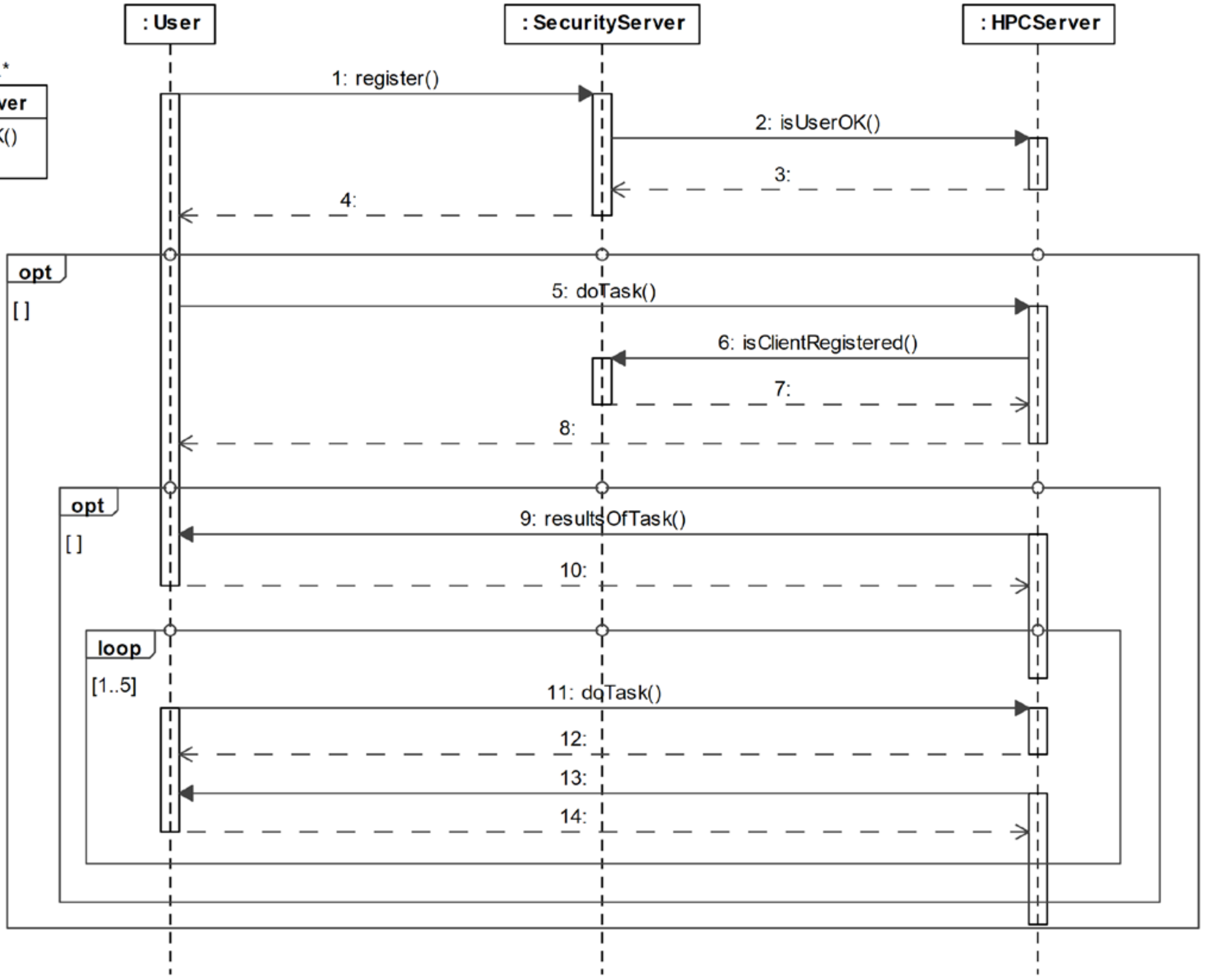
- Performance degrades by the use of nested loops or loops with nested operators
- To solve the problem, we decided to process the content of each loop as a separate MSC and store its content in files.
- These files would be repeatedly written in the output files.
- The algorithm takes seconds instead of hours.



Validation…



SecurityServer
+register()
+isClientRegistered()
User
+resultsOfTask()
HPCServer
+isUserOK()
+doTask()
1..*
0..*
1
0..*
0..*
1..*



: User
: SecurityServer
: HPCServer
1: register()
2: isUserOK()
3:
4:
opt
[ ]
5: doTask()
6: isClientRegistered()
7:
8:
opt
[ ]
9: resultsOfTask()
10:
loop
[1..5]
11: doTask()
12:
13:
14:

Future work

- Support more combined operators such as seq, critical or beak.
- Add guards to the combined fragments (this is currently abstracted away)
- Embedding the plugin to USE environment.
- Validate the plugin with larger case studies and study its performance.
- Analyze the usability of the proposal, conducting empirical experiments with real modelers

Automatic Generation of Valid Behavioral Scripts from UML Sequence Diagrams

Paula Muñoz, Loli Burgueño, Antonio Vallecillo, and Martin Gogolla

OCL and Textual Modeling Workshop @ MODELS 2019

Munich, Germany, September 16, 2019