MODELING AND GENERATION OF TEST CASES BASED ON SEQUENCE DIAGRAMS

Alexandre Petrenko
El Hachemi Alikacem
Computer Research Institute of Montreal, CRIM

ERICSSON MODELING DAYS
September 13-14, 2016

Recent Project Ericsson-CRIM

- This project was led by Francis Bordeleau
  in collaboration with Edgard Fiallos, and Norman Dack, Ericsson (Ottawa) involved in testing telephony systems
- Tools were developed by El Hachemi Alikacem, CRIM
- The project had a limited support and resulted in a prototype tool as a proof-of-concept

Test Scenario Modeling

A test scenario is represented by a sequence diagram derived from system's specification, use cases, design models or legacy test cases

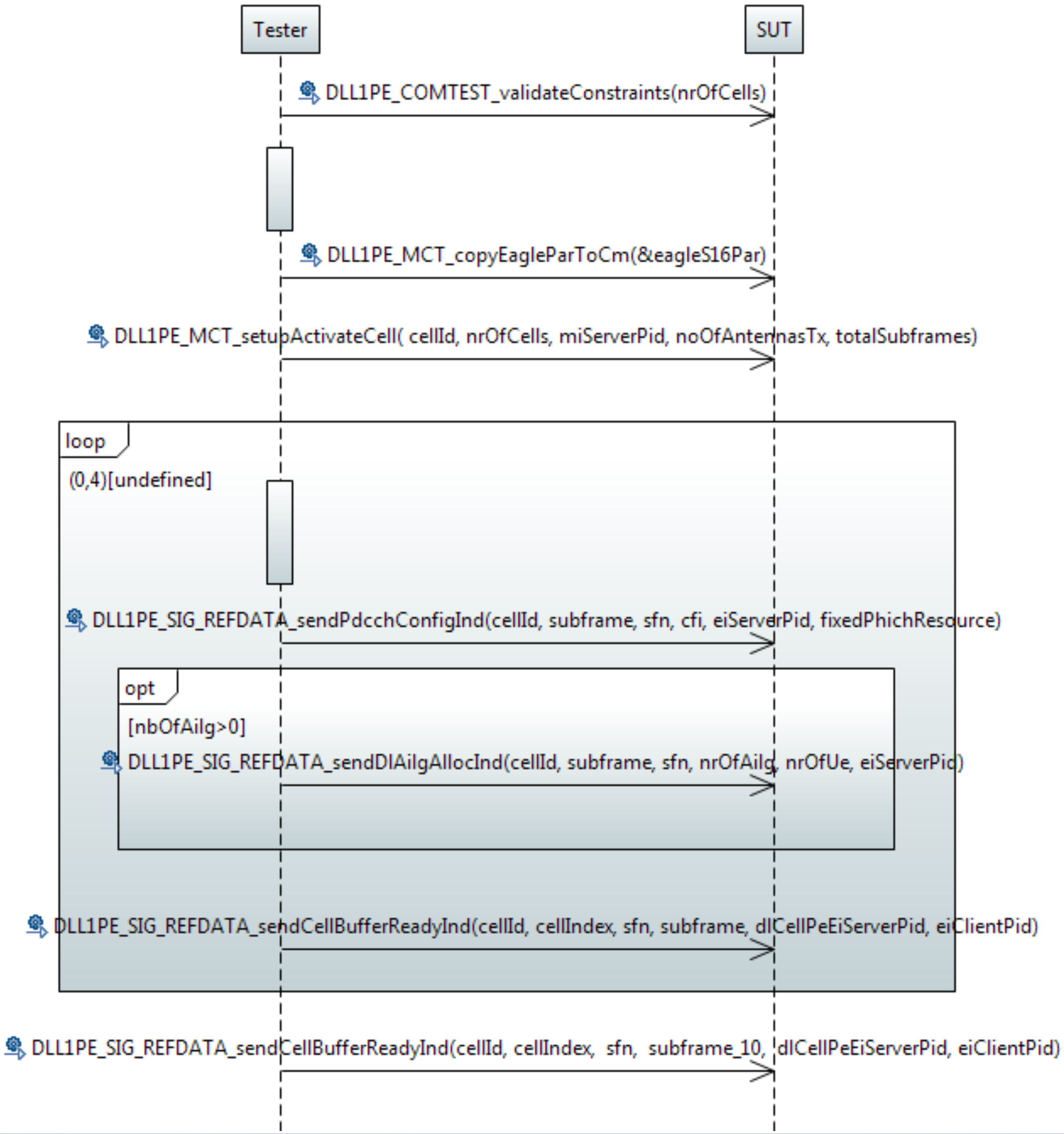
- One lifeline could be chosen as a future tester
- Several testers/test stubs need coordination
- The remaining lifelines represent the System Under Test (SUT) components

A right model to automate integration/system testing?

Sequence Diagram Features

- Lifelines: single tester, one or several SUT components
- Asynchronous Abstract Messages
- Alternative Blocks
- Option Blocks
- Parallel Blocks
- Nested Blocks
- Co-Regions
- Loops, while loop included
- Delays
- Local actions, code snippets

Test Scenario Modelling a Simplified Ericsson Test



CRIM
Votre longueur d'avance

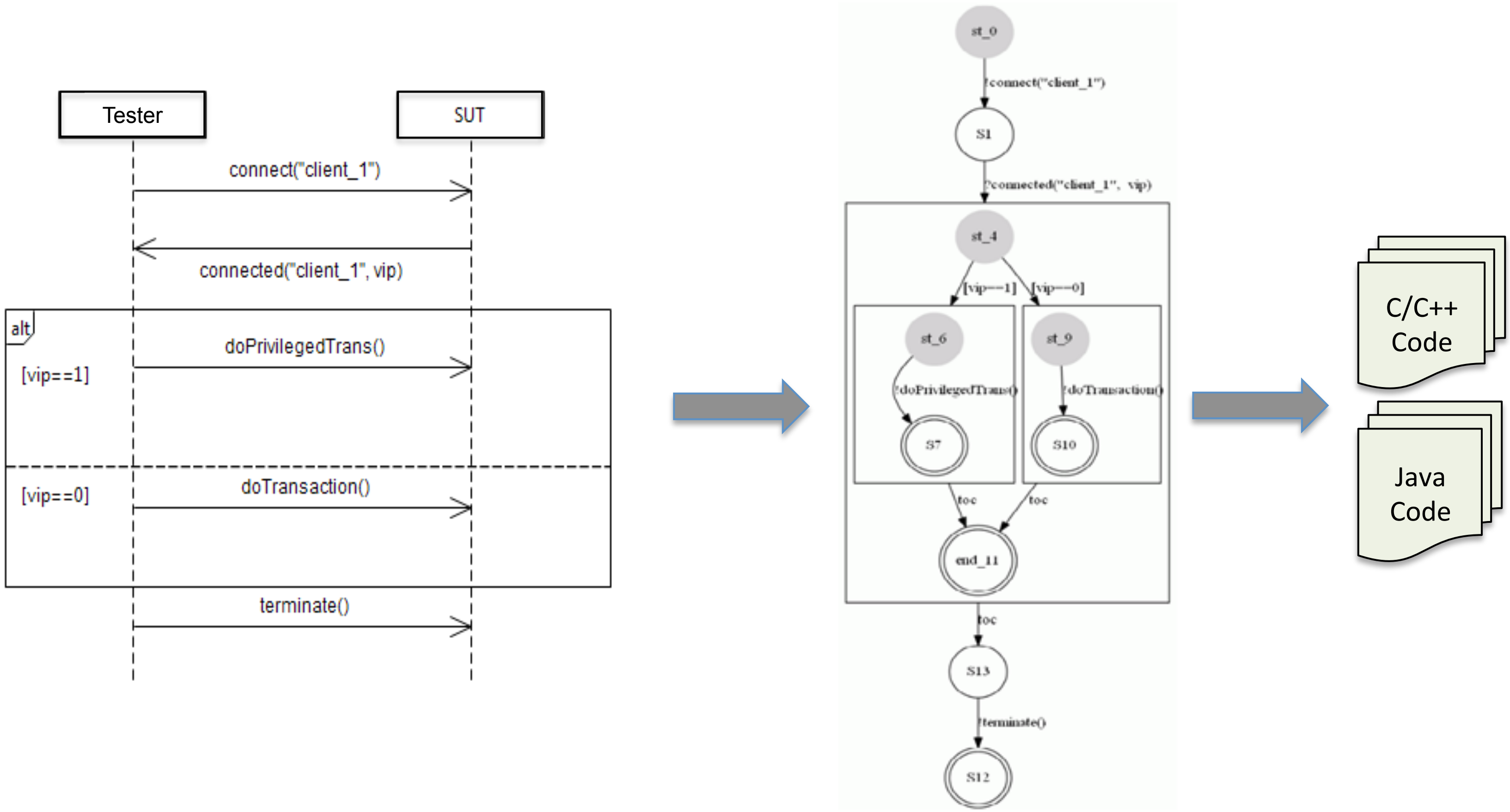
Using Test Scenarios

- Executable tester generation

- Test scenario simulation
  - Validate test scenario
  - Collect traces
  - Estimate fault detection by mutating the simulated SUT

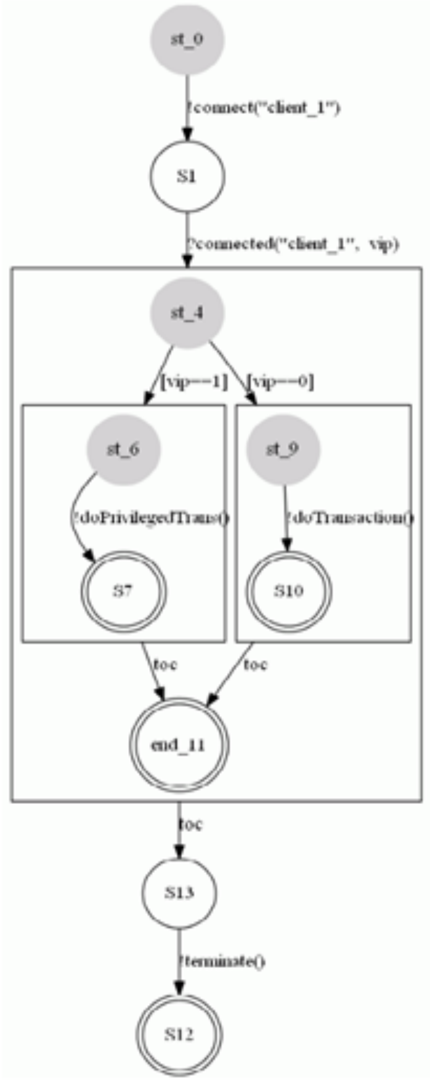
Test Generation Approach

Via a sequence diagram to statechart transformation

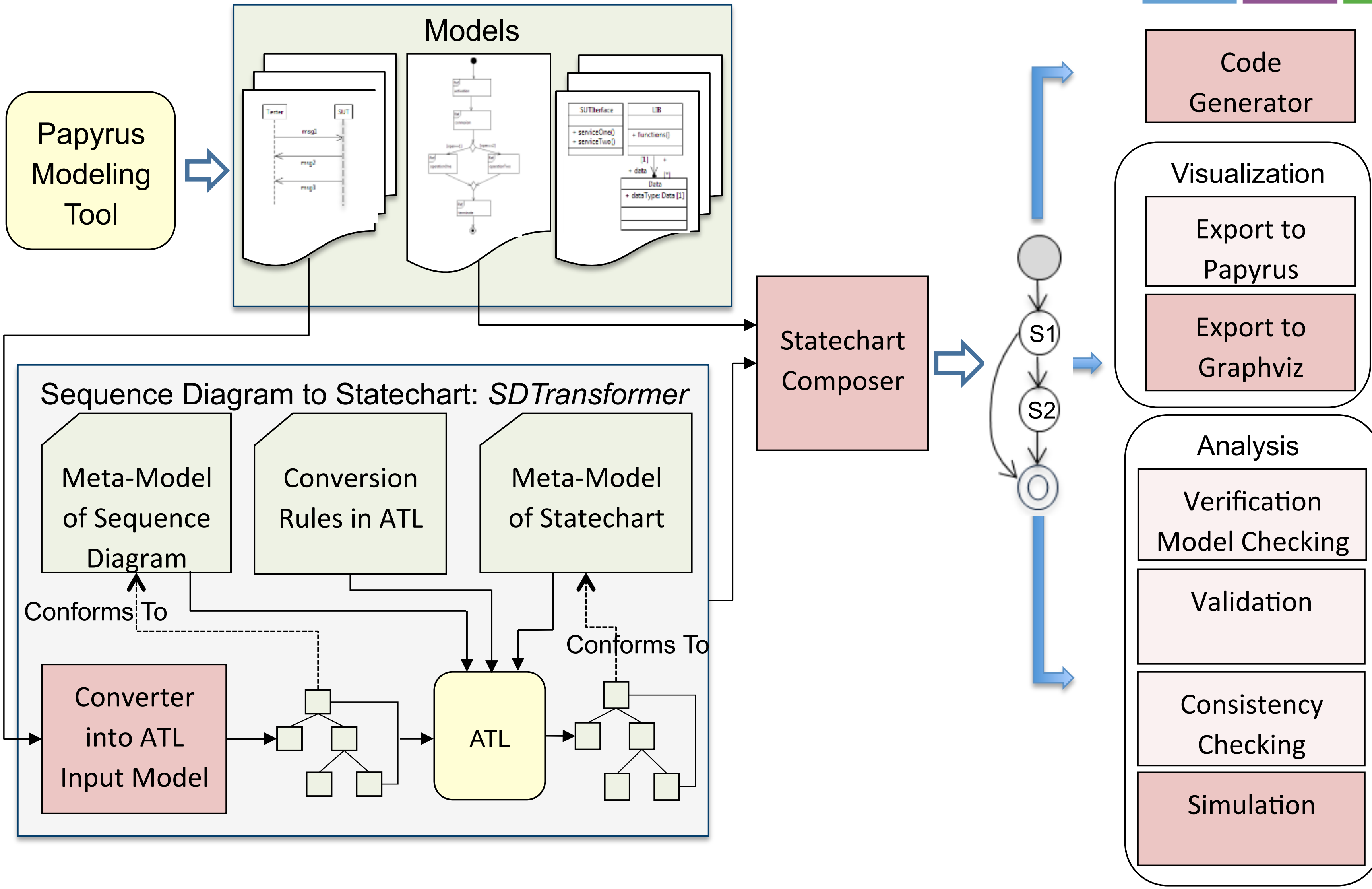


Hierarchical Statechart

- A Hierarchical statechart [Harel, 1987] is a finite state machine (FSM) with composite states refined by other FSMs
- Statecharts (and their numerous variants) are widely used to model behavior, and many tools support them
- Some tools provide statecharts analysis, property verification, model checking, simulation, etc.



Our Framework



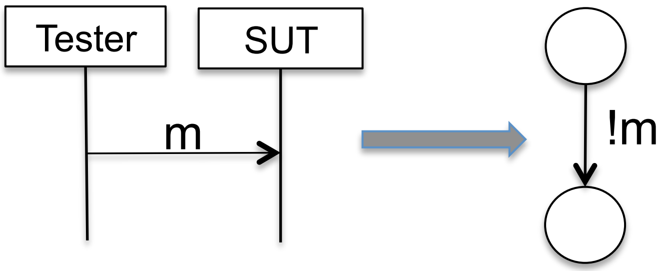
SDTransformer Features

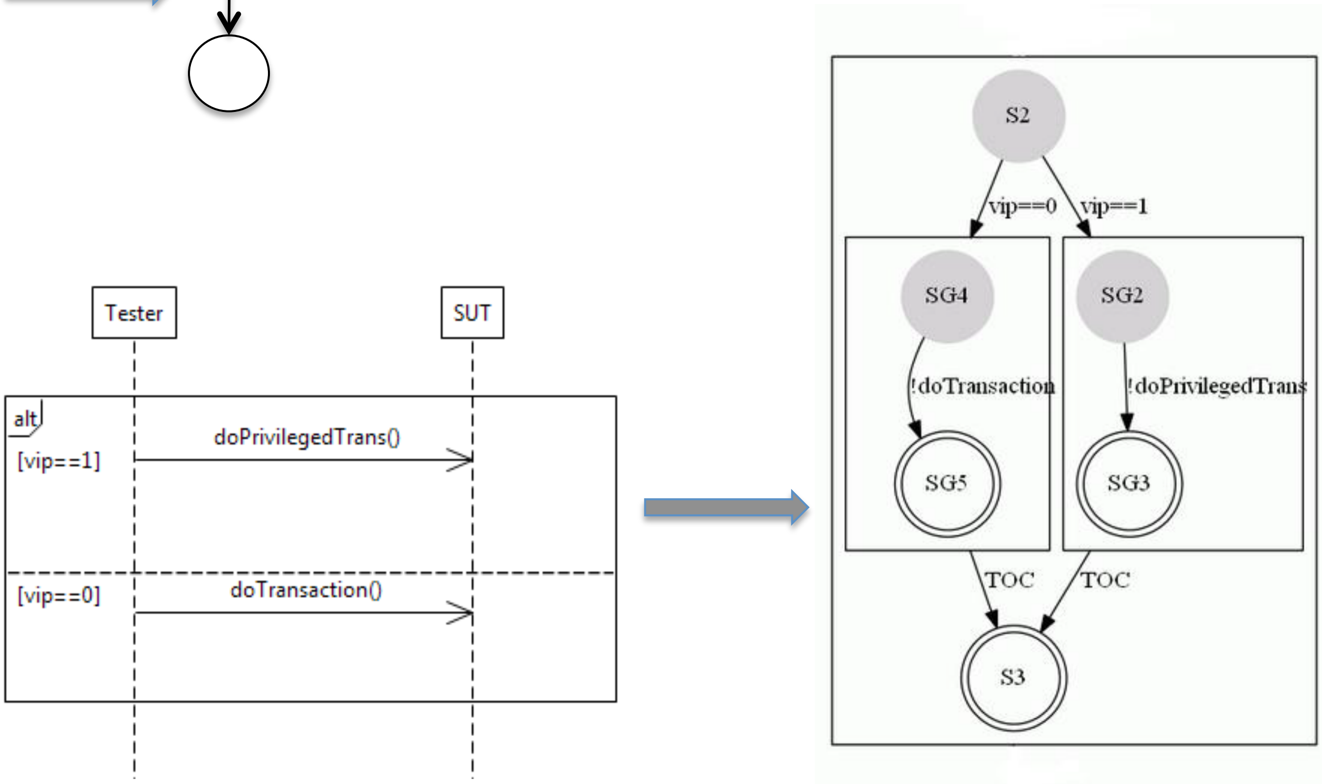
- Sequence Diagram to Statechart Transformation
- Statechart Visualization
- Composing Test Scenarios from a Graph of Test Scenarios
- Executable Test Case Generation
- Test Scenario Simulation

SDTransformer is an Eclipse plugin integrated with Papyrus modeling tool

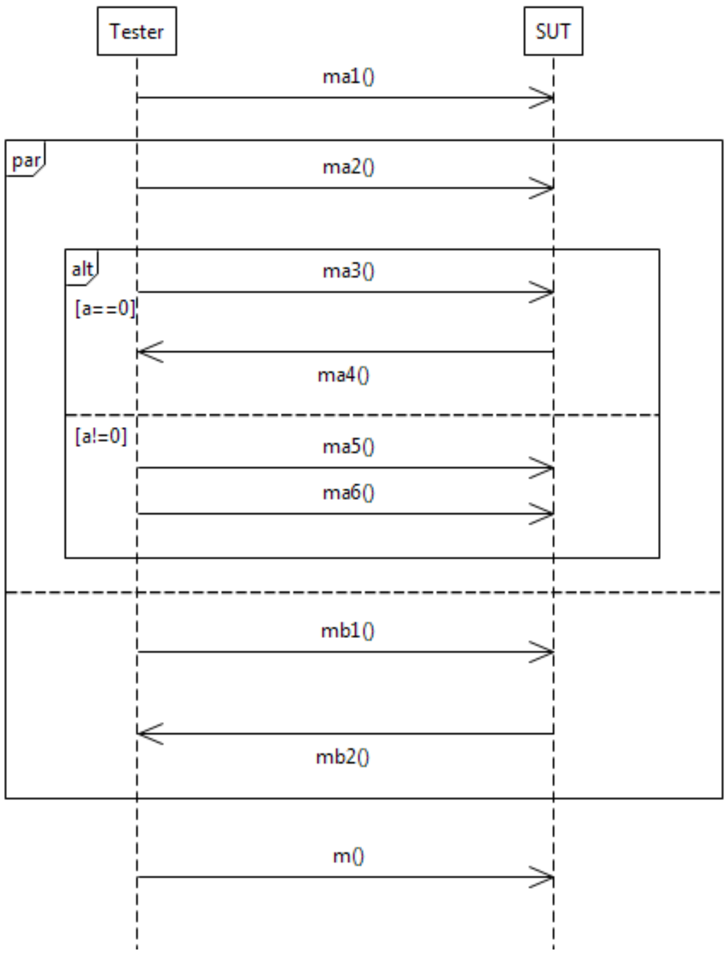
Model Transformation

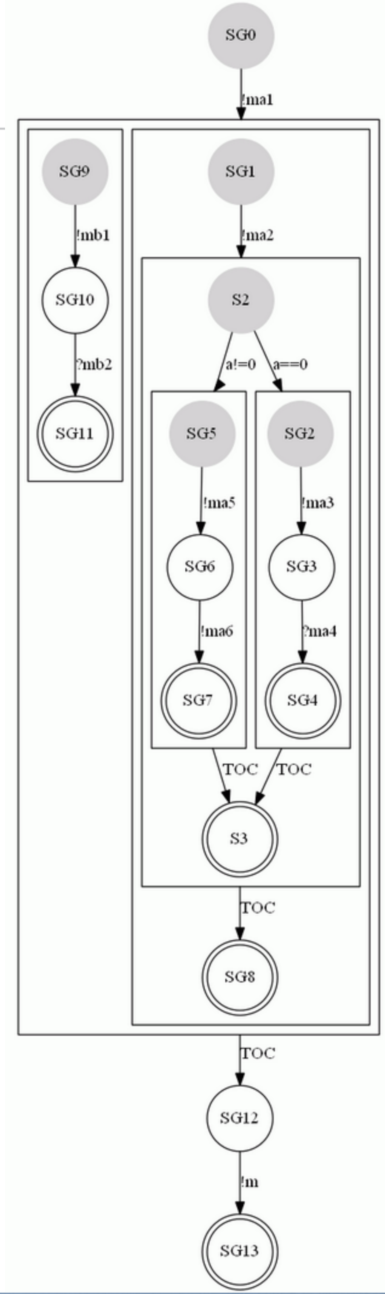
A transformation algorithm is implemented using ATL model transformation tool



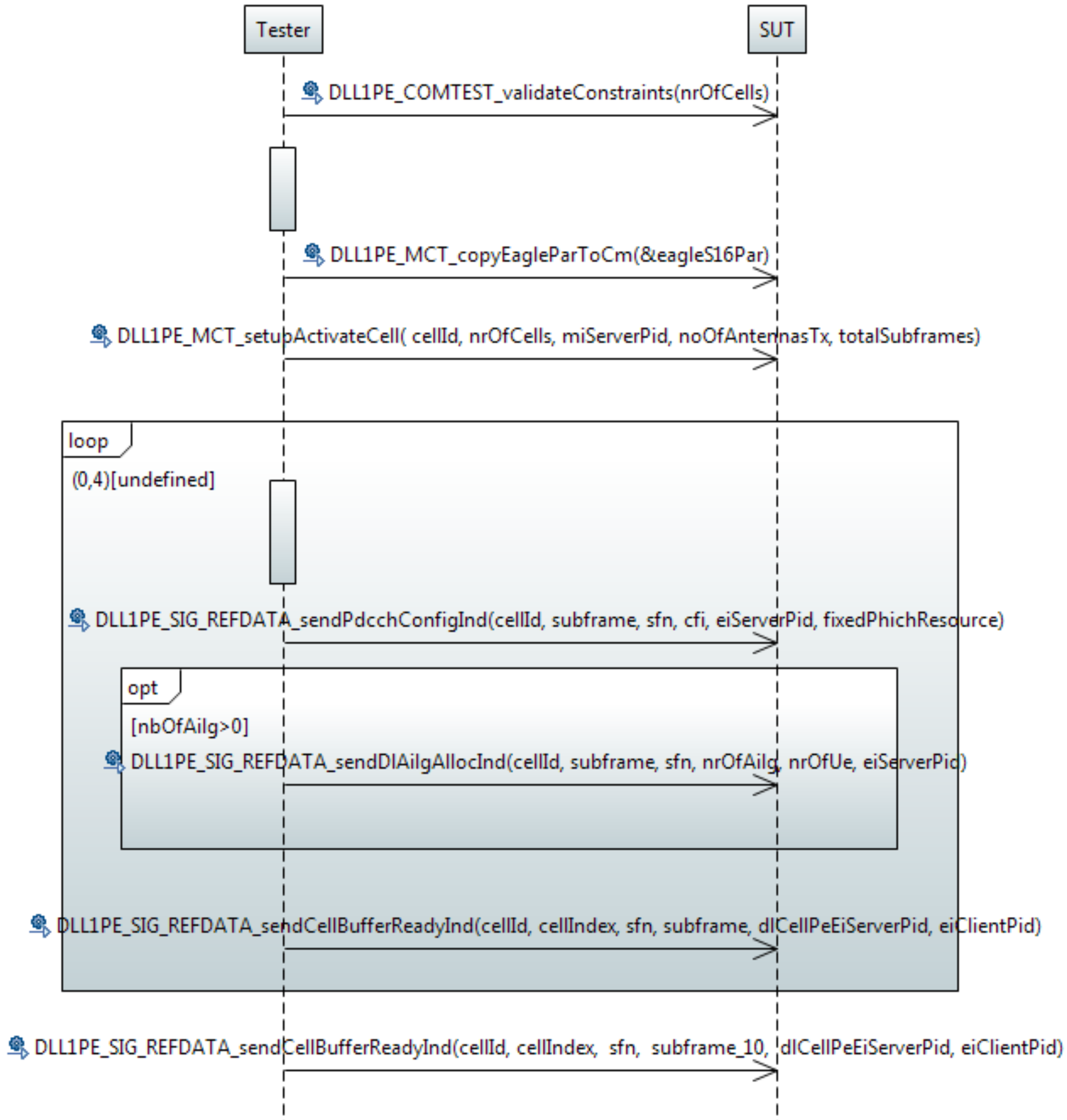


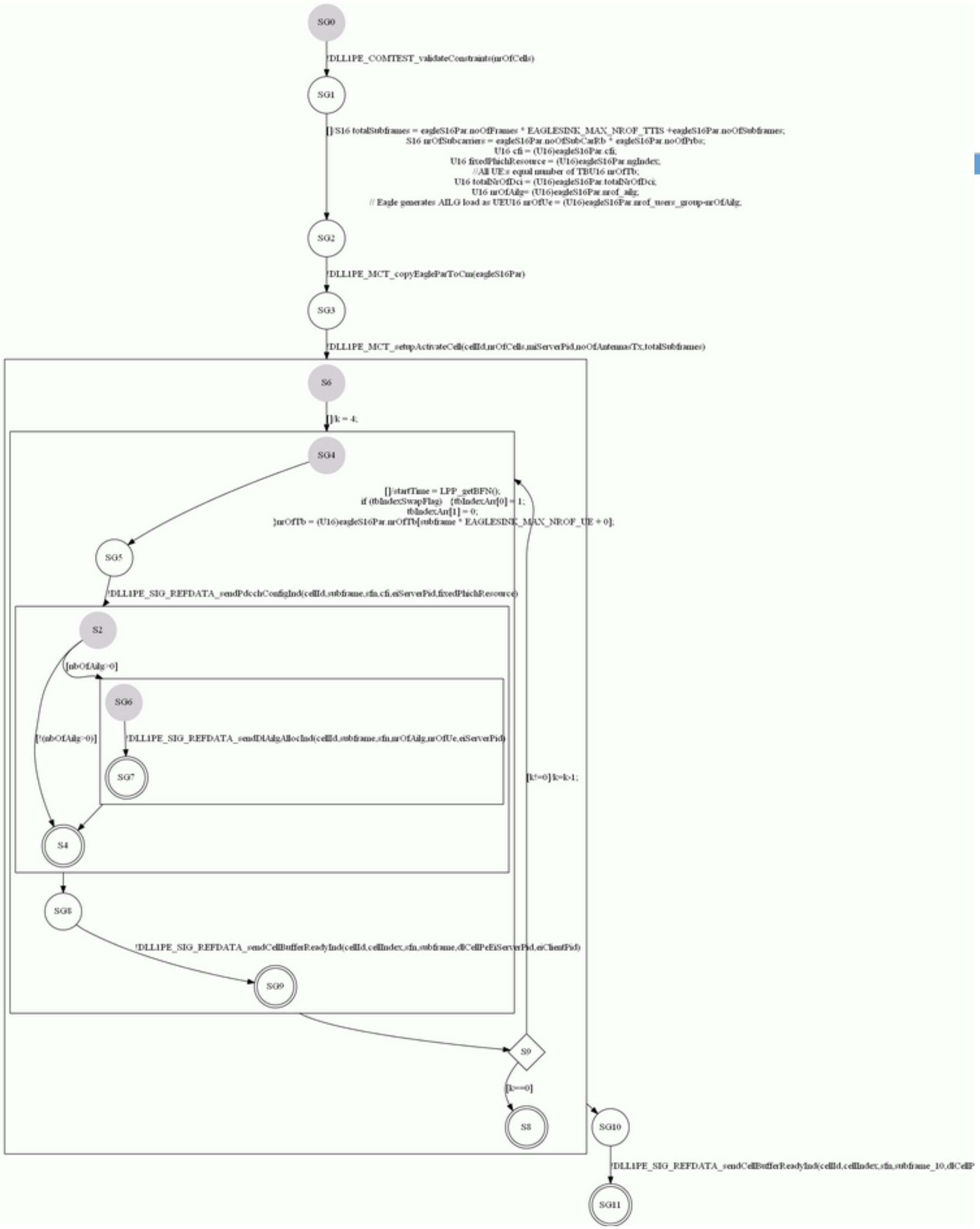
Nested Blocks





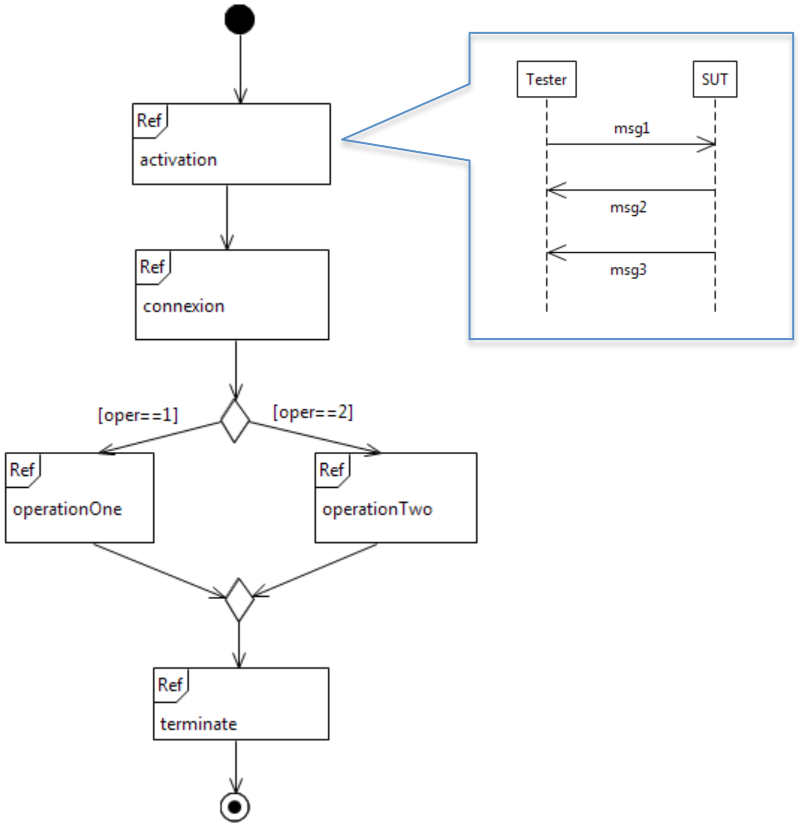
Test Scenario Modelling a Simplified Ericsson Test




SG0
!DLL1PE_COMTEST_validateConstraints(nrOfCells)
SG1
[]/S16 totalSubframes = eagleS16Par.noOfFrames * EAGLESINK_MAX_NROF_TTIS +eagleS16Par.noOfSubframes;
S16 nrOfSubcarriers = eagleS16Par.noOfSubCarRb * eagleS16Par.noOfPrbs;
U16 cfi = (U16)eagleS16Par.cfi;
U16 fixedPhichResource = (U16)eagleS16Par.ngIndex;
//All UE:s equal number of TBU16 nrOfTb;
U16 totalNrOfDci = (U16)eagleS16Par.totalNrOfDci;
U16 nrOfAilg= (U16)eagleS16Par.nrof_ailg;
// Eagle generates AILG load as UEU16 nrOfUe = (U16)eagleS16Par.nrof_users_group-nrOfAilg;
SG2
!DLL1PE_MCT_copyEagleParToCm(eagleS16Par)
SG3
!DLL1PE_MCT_setupActivateCell(cellId,nrOfCells,miServerPid,noOfAntennasTx,totalSubframes)
S6
[]/k = 4;
SG4
[]/startTime = LPP_getBFN();
if (tbIndexSwapFlag) {tbIndexArr[0] = 1;
tbIndexArr[1] = 0;
}nrOfTb = (U16)eagleS16Par.nrOfTb[subframe * EAGLESINK_MAX_NROF_UE + 0];
SG5
!DLL1PE_SIG_REFDATA_sendPdcchConfigInd(cellId,subframe,sfn,cfi,eiServerPid,fixedPhichResource)
S2
[nbOfAilg>0]
SG6
!DLL1PE_SIG_REFDATA_sendDlAilgAllocInd(cellId,subframe,sfn,nrOfAilg,nrOfUe,eiServerPid)
SG7
[!(nbOfAilg>0)]
S4
SG8
!DLL1PE_SIG_REFDATA_sendCellBufferReadyInd(cellId,cellIndex,sfn,subframe,dlCellPeEiServerPid,eiClientPid)
SG9
[k!=0]/k=k-1;
S9
[k==0]
S8
SG10
!DLL1PE_SIG_REFDATA_sendCellBufferReadyInd(cellId,cellIndex,sfn,subframe_10,dlCellP
SG11

Graph of Test Scenarios

Interaction Diagram is used to specify a Graph of Test Scenario, where a reference node represents an Atomic Test Scenario

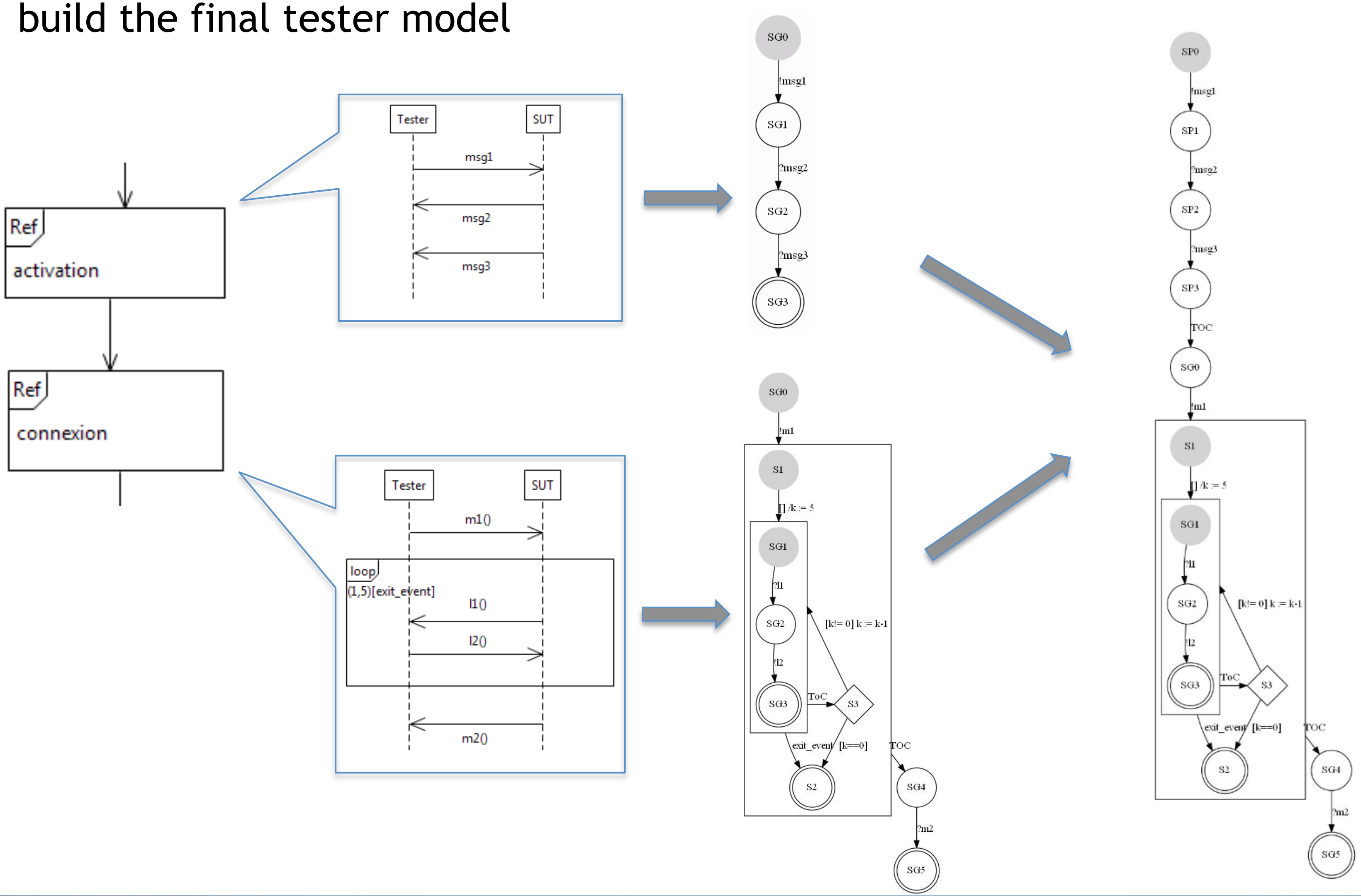


Interaction Diagram Features

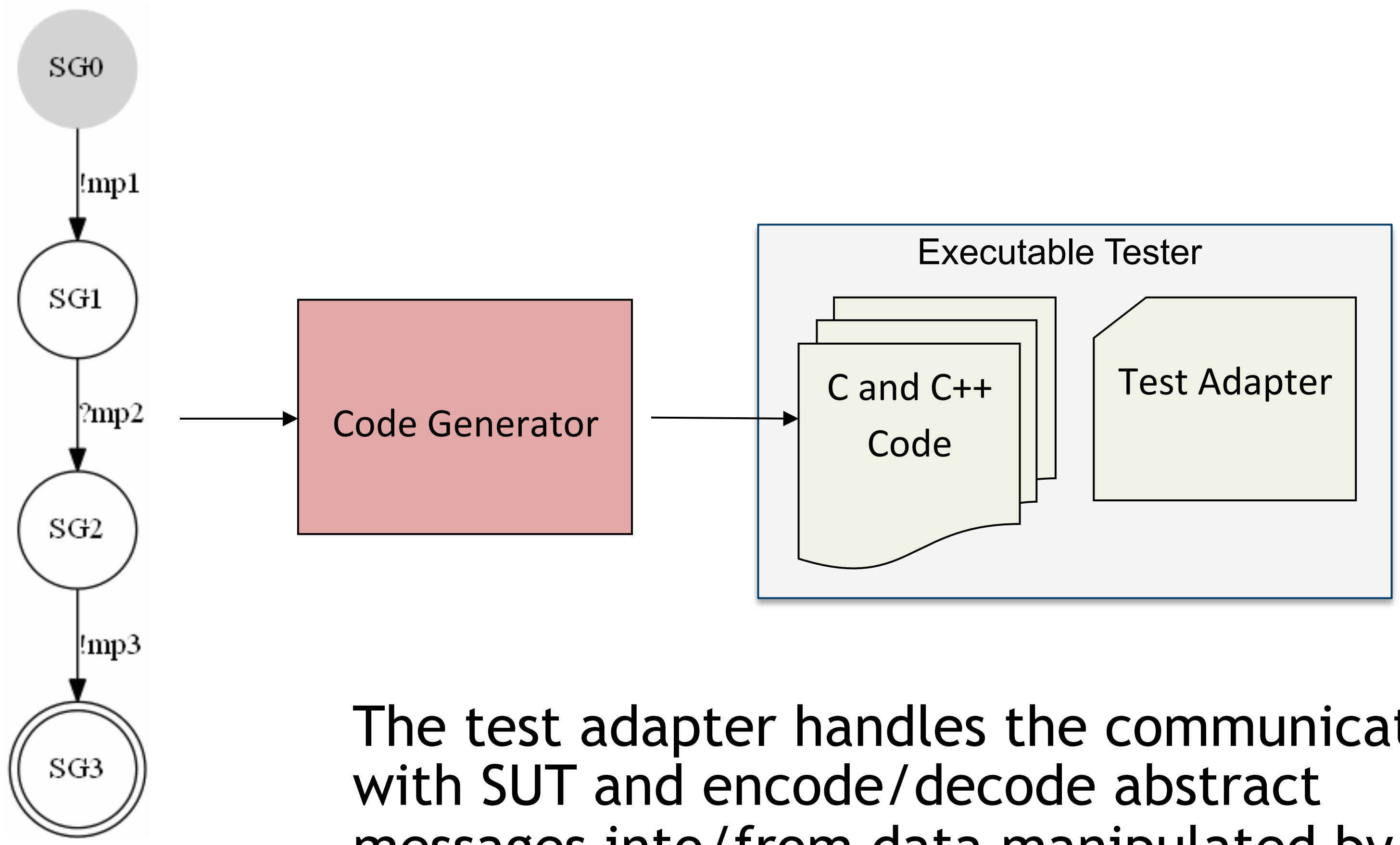
- Initial Node and Final Node
- Reference Node
- Decision Node and Merge Node
- Guards

Example

Statecharts generated from each atomic test scenario are composed to build the final tester model



Executable Tester Generation



The test adapter handles the communication with SUT and encode/decode abstract messages into/from data manipulated by the SUT

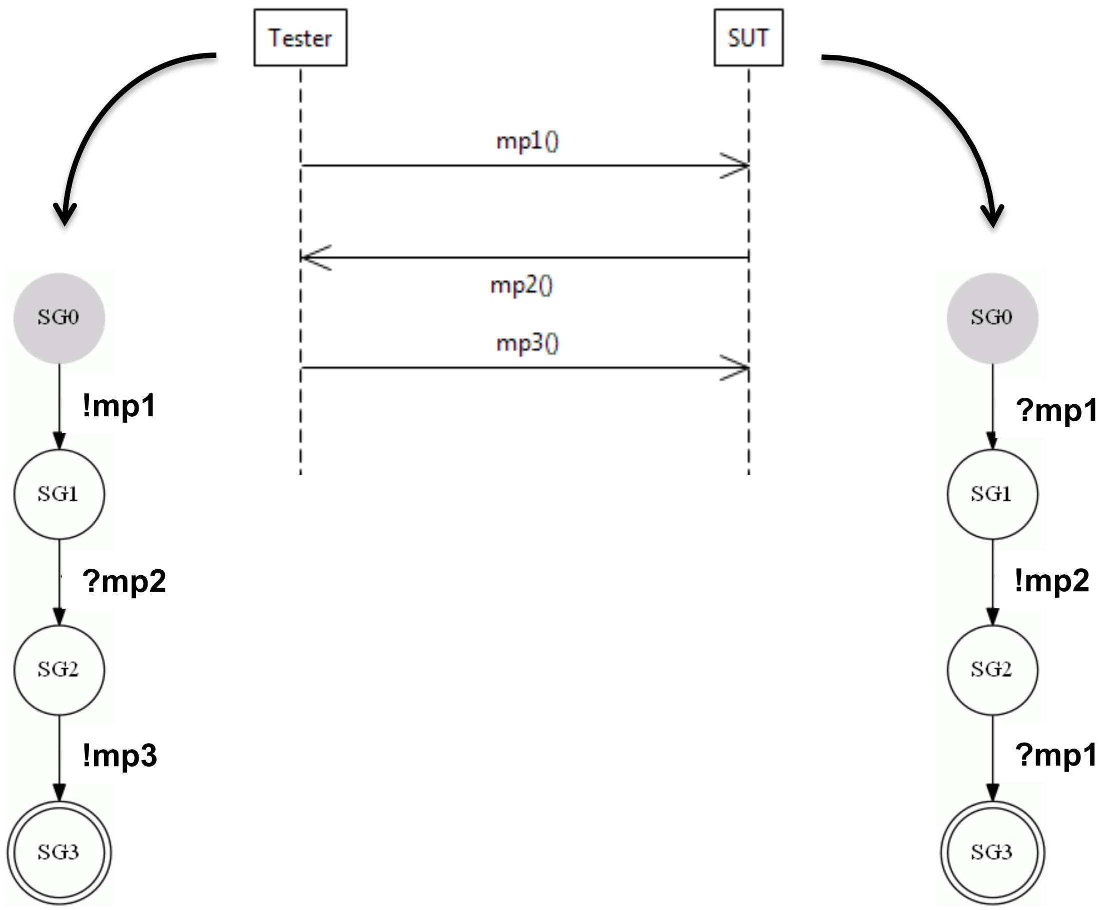
It is SUT-dependent (sockets, CAN Bus, …)

Using Test Scenarios

- Executable tester generation
- Test scenario simulation
  - Validate test scenario
  - Collect traces
  - Estimate fault detection by mutating the simulated SUT

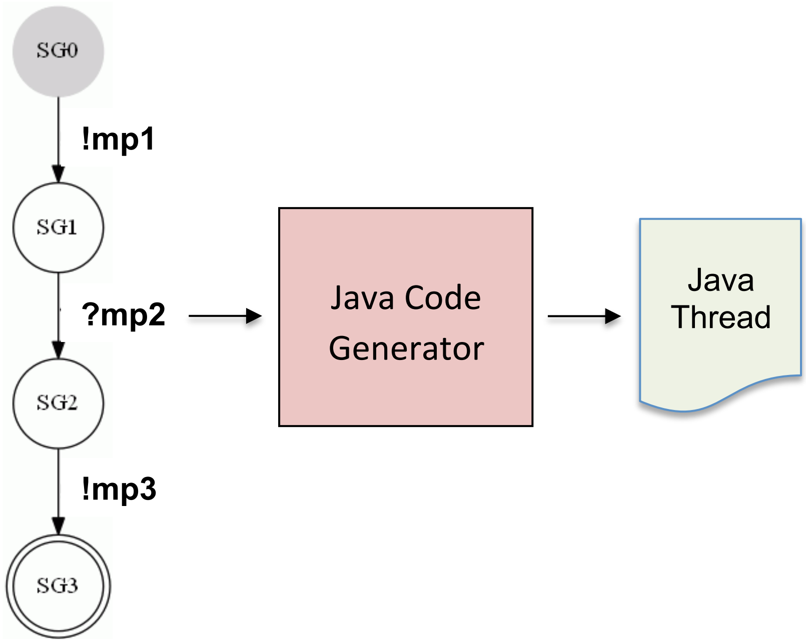
Building Simulated System in Java, Step 1

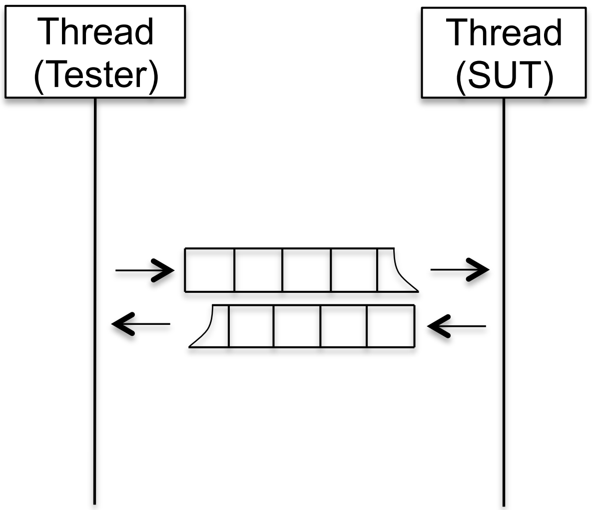
A statechart is generated for each lifeline of the scenario



Building Simulated System in Java, Step 2

Generation of a Java thread for each statechart





For each pair of threads, we define two blocking queues

Perspective

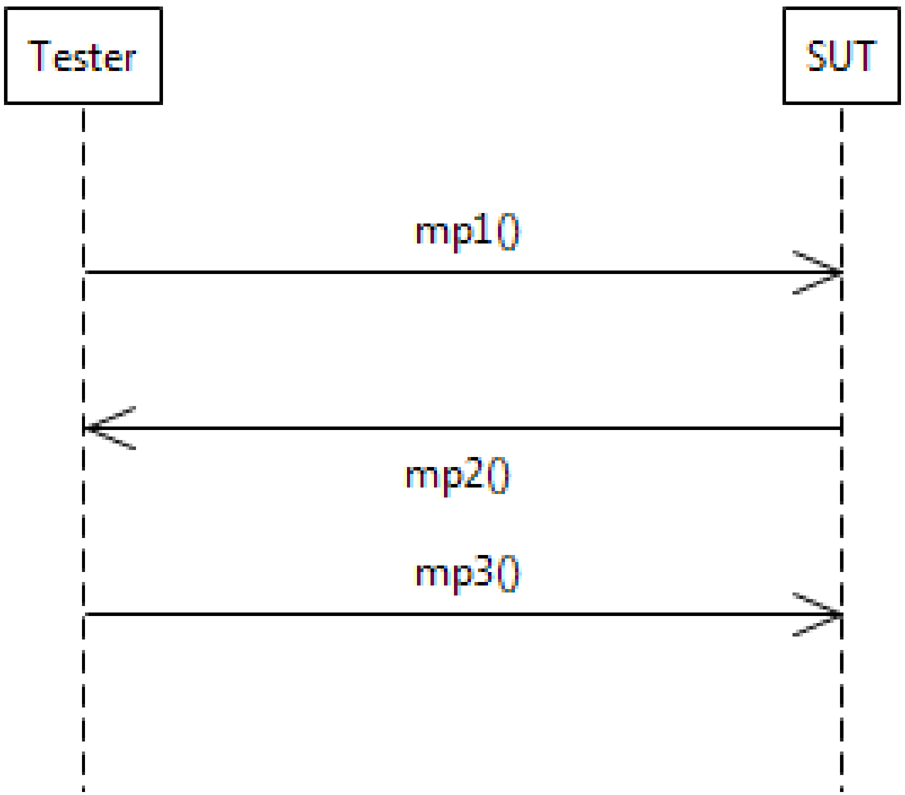
- SDTransformer extensions
  - Supporting additional features in a sequence diagram, such as parameters, time, variables
  - Handling test adapters
- Test Scenario Verification

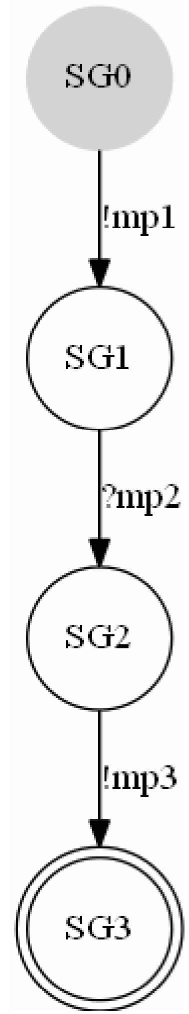
Conclusions

- We were exposed to some test automation problems of one team of Ericsson
- The usefulness of the suggested approach for that team was demonstrated
- We are open for more collaborations on MBT with sequence diagrams and other models

thank you very much

Example





Code Generation: Main Functions

```
int MainRegion_runCycle(MainRegion_context *scExecution_ctx) {
    int res = -1 ;
    switch(scExecution_ctx->currentStateLabel) {
        case MainRegion_InitialState_SG0 :
            process_MainRegion_InitialState_SG0(scExecution_ctx) ;
            break ;
        case MainRegion_State_SG1 :
            process_MainRegion_State_SG1(scExecution_ctx) ;
            break ;
        case MainRegion_State_SG2 :
            process_MainRegion_State_SG2(scExecution_ctx) ;
            break ;
        case MainRegion_FinalState_SG3 :
            process_MainRegion_FinalState_SG3(scExecution_ctx) ;
            break ;
        default :
            break ;
    } // switch
    return res;
}

int MainRegion_machineExcecution() {
    codecRef=new OperTwo_CODEC();
    MainRegion_context *scExecution_ctx= new MainRegion_context();
    scExecution_ctx->currentStateLabel=MainRegion_InitialState_SG0;

    while (scExecution_ctx->terminated!= 1) {
        MainRegion_runCycle(scExecution_ctx);
    }
}
```

Code Generation: State Processing

```
/* Processing State  SG0 */
void process_MainRegion_InitialState_SG0(MainRegion_context *scExecution_ctx) {
   if (__Trace==1) {
         cout<<"[Trace] Entering State: MainRegion_InitialState_SG0" << endl;
   }
   mp1_transition(scExecution_ctx);
}

/* Processing State  SG1 */
void process_MainRegion_State_SG1(MainRegion_context *scExecution_ctx) {
 if (__Trace==1) {
       cout<<"[Trace] Entering State: MainRegion_State_SG1" << endl;
   }
   mp2_transition(scExecution_ctx);
}

/* Processing State  SG2 */
void process_MainRegion_State_SG2(MainRegion_context *scExecution_ctx) {
 if (__Trace==1) {
       cout<<"[Trace] Entering State: MainRegion_State_SG2" << endl;
   }
   mp3_transition(scExecution_ctx);
}

/* Processing State  SG3 */
void process_MainRegion_FinalState_SG3(MainRegion_context *scExecution_ctx) {
 if (__Trace==1) {
       cout<<"[Trace] Entering State: MainRegion_FinalState_SG3" << endl;
   }
   scExecution_ctx->terminated=1;
}
```

Code Generation: Transition Processing

```
/* Processing Transition FROM: SG0  TO: SG1 */
int mp1_transition(MainRegion_context *scExecution_ctx) {
 if (__Trace==1) {
       cout<<"[Trace] Processing Transition : mp1" << endl;
   }
   codecRef->mp1_SUTSEND();
   scExecution_ctx->currentStateLabel = MainRegion_State_SG1;
   return 1;
}

/* Processing Transition FROM: SG1  TO: SG2  */
int mp2_transition(MainRegion_context *scExecution_ctx) {
 if (__Trace==1) {
       cout<<"[Trace] Processing Transition : mp2" << endl;
   }
   codecRef->mp2_SUTRECEIVE();
   scExecution_ctx->currentStateLabel = MainRegion_State_SG2;
   return 1;
}

/* Processing Transition FROM: SG2  TO: SG3 */
int mp3_transition(MainRegion_context *scExecution_ctx) {
 if (__Trace==1) {
       cout<<"[Trace] Processing Transition : mp3" << endl;
   }
   codecRef->mp3_SUTSEND();
   scExecution_ctx->currentStateLabel = MainRegion_FinalState_SG3;
    return 1;
}
```