Detecting Vulnerabilities in Java-Card Bytecode Verifiers using Model-Based Testing

Aymerick Savary[1,2] Marc Frappier[1] Jean-Louis Lanet[2]

[1]Université de Sherbrooke

[2]Université de Limoges

13-06-2013

Outline

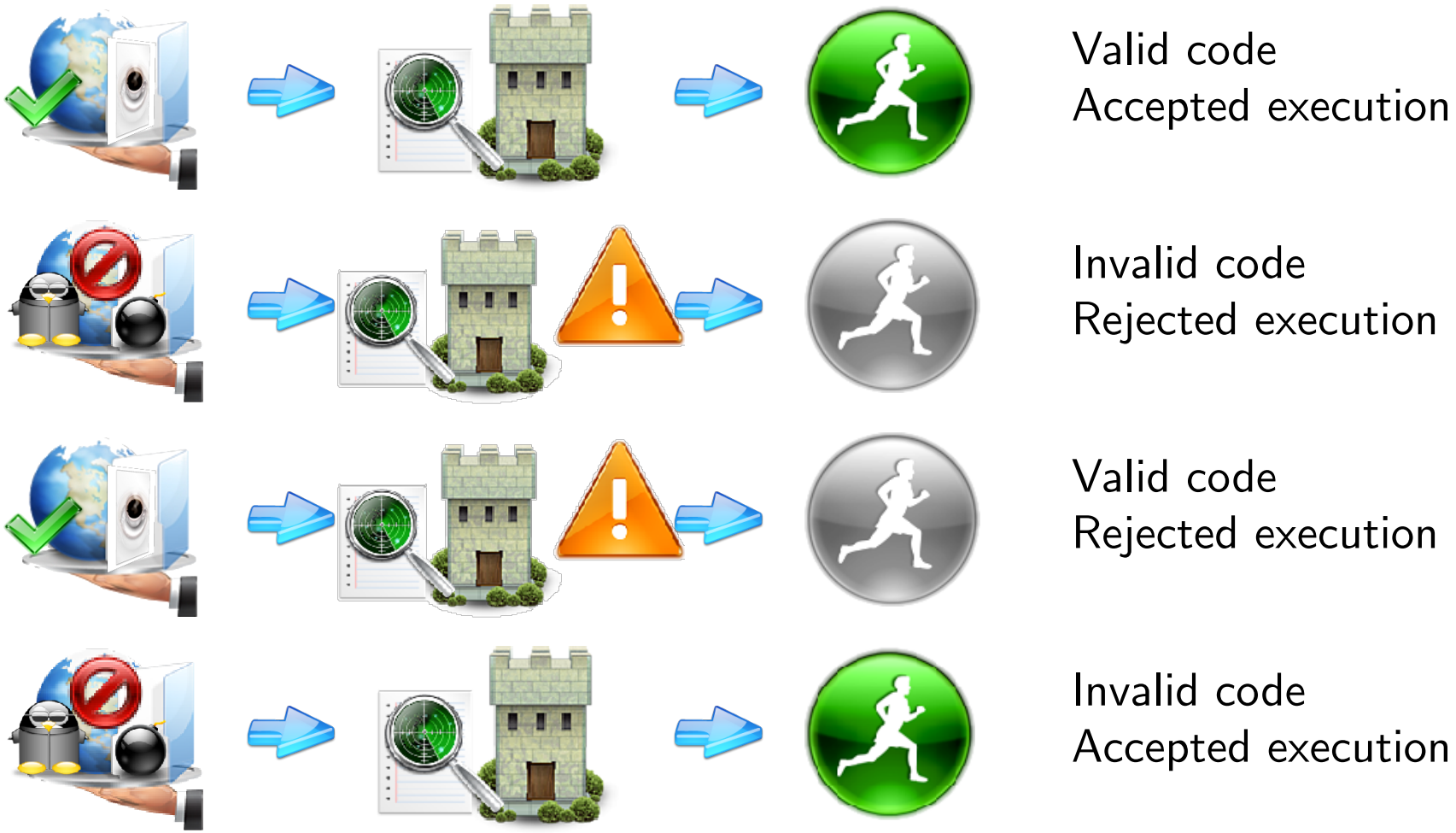
Java Card Byte Code Verifier

Byte Code verifier behavior

Valid code
Accepted execution

Invalid code
Rejected execution

Byte Code verifier behavior



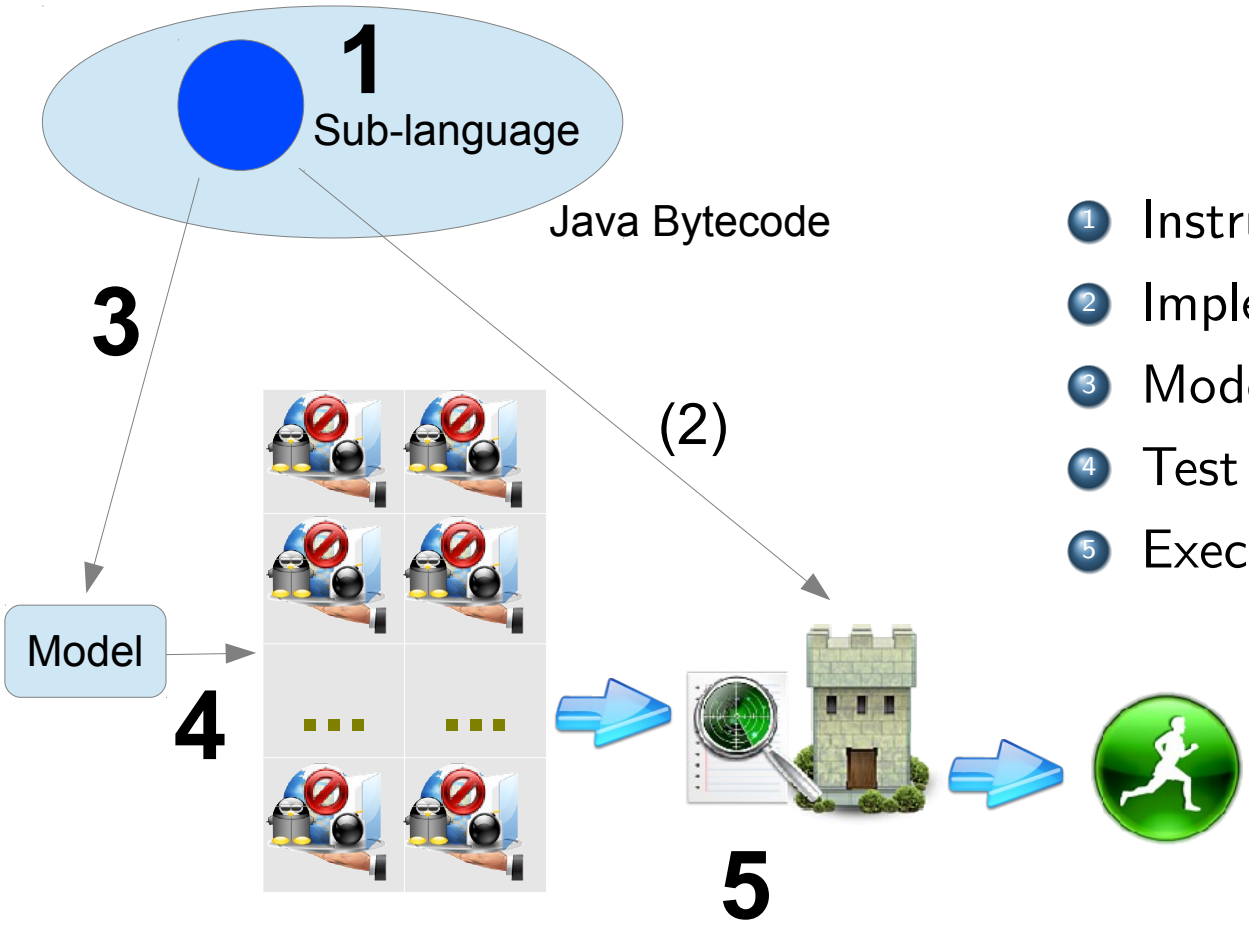
Valid code
Accepted execution

Invalid code
Rejected execution

Valid code
Rejected execution

Invalid code
Accepted execution

Step of the project



1. Instruction subset
2. Implement BCV
3. Modeling
4. Test generation
5. Execution

Algorithm: Guards Grouping

EVENTS
Event $sload \mathrel{\widehat{=}}$

any

$\mathtt{index}$

where

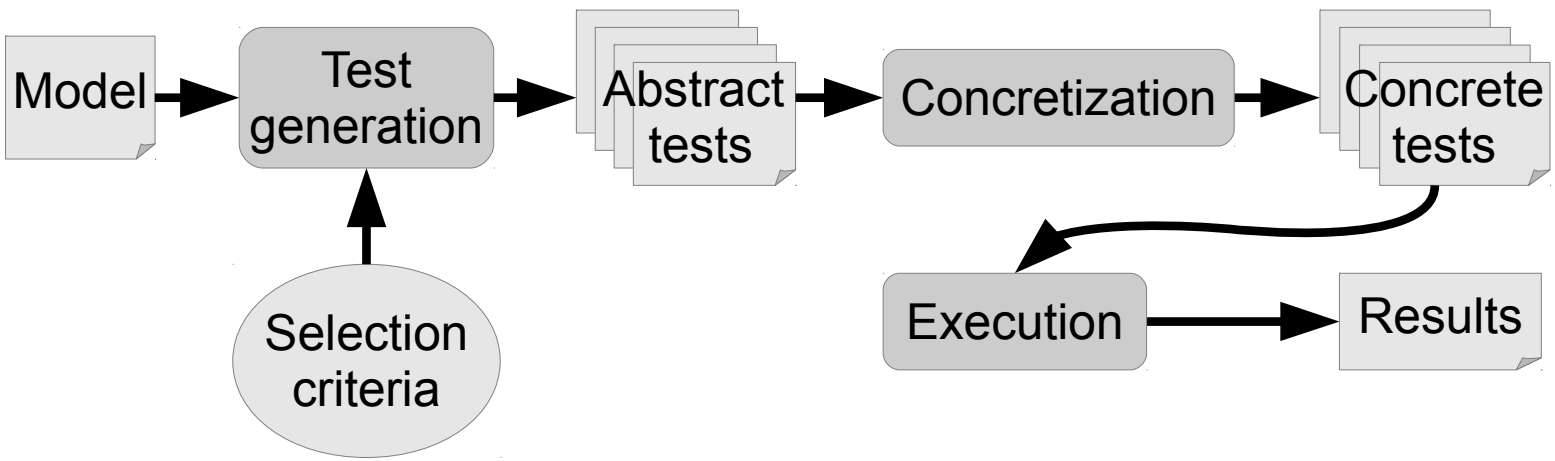
$\mathtt{grd1} : \mathtt{halt} = \mathtt{FALSE}$
$\mathtt{grd2} : \mathtt{pc} < \mathtt{maxpc}$
$\mathtt{grd1\_t} : \mathtt{z}(\mathtt{pc}) \leq \mathtt{maxstack} - 1$
$\mathtt{grd2\_t} : \mathtt{index} \in 1 \mathinner{..} \mathtt{maxlocalvar}$
$\mathtt{grd3\_t} : \mathtt{v}(\mathtt{pc})(\mathtt{index}) = \mathtt{short}$

then

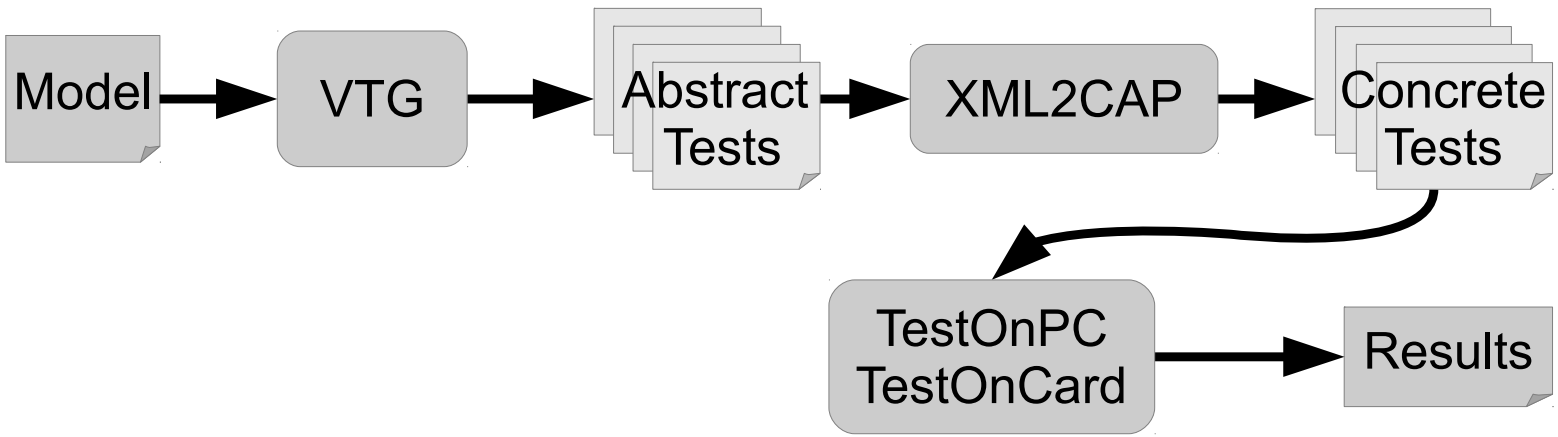
$\mathtt{act1} : \mathtt{pc} := \mathtt{pc} + 1$
$\mathtt{act2} : \mathtt{s} := \mathtt{s} \lhd \{\mathtt{pc} + 1 \mapsto \mathtt{s}(\mathtt{pc}) \lhd \{\mathtt{z}(\mathtt{pc}) \mapsto \mathtt{short}\}\}$
$\mathtt{act3} : \mathtt{z} := \mathtt{z} \lhd \{\mathtt{pc} + 1 \mapsto \mathtt{z}(\mathtt{pc}) + 1\}$
$\mathtt{act4} : \mathtt{v} := \mathtt{v} \lhd \{\mathtt{pc} + 1 \mapsto \mathtt{v}(\mathtt{pc})\}$

end

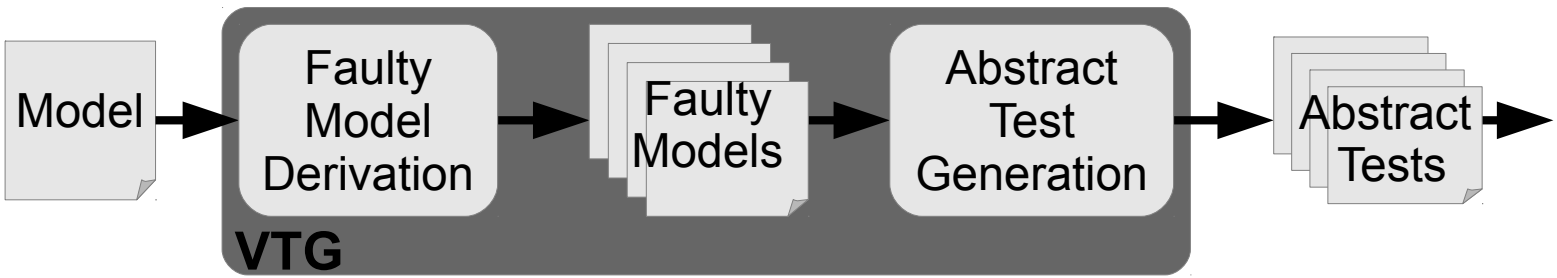
Model Based Testing Approach



Big Picture


Model
VTG
Abstract Tests
XML2CAP
Concrete Tests
TestOnPC TestOnCard
Results

VTG process


Model
Faulty Model Derivation
Faulty Models
Abstract Test Generation
Abstract Tests
VTG

Algorithm

Input: m : *Event-B* model
Output: M' : set of *Event-B* model
for *each event* e *of* m **do**
  rewrite the guard of e into two guards:
    grd, the conjunction of all guards of e without suffix "_t";
    grd_t, the conjunction of all guards of e with suffix "_t";
end
for *each event* e *of* m **do**
  $e.RW := neg(grd\_t)$;
end
for *each event* e *of* m **do**
  for *each* rw *in* $e.RW$ **do**
    add a new model m' to M' such that
      $m' := m$;
      $m'.events := m'.events \cup \{e'\}$, where e' is defined as follows:
        $e' := e$;
        replace $e'.grd\_t$ by rw;
        add guard "`eut = FALSE`" to e';
        add action "`eut := TRUE`" to e';
  end
end

Algorithm 1: Faulty model derivation algorithm

Algorithm: Guards Grouping

EVENTS
Event $sload \mathrel{\widehat{=}}$

any

$\mathtt{index}$

where

$\mathtt{grd1} : \mathtt{halt} = \mathtt{FALSE}$
$\mathtt{grd2} : \mathtt{pc} < \mathtt{maxpc}$
$\mathtt{grd1\_t} : \mathtt{z}(\mathtt{pc}) \leq \mathtt{maxstack} - 1$
$\mathtt{grd2\_t} : \mathtt{index} \in 1 \mathinner{..} \mathtt{maxlocalvar}$
$\mathtt{grd3\_t} : \mathtt{v}(\mathtt{pc})(\mathtt{index}) = \mathtt{short}$

then

$\mathtt{act1} : \mathtt{pc} := \mathtt{pc} + 1$
$\mathtt{act2} : \mathtt{s} := \mathtt{s} \lhd \{\mathtt{pc} + 1 \mapsto \mathtt{s}(\mathtt{pc}) \lhd \{\mathtt{z}(\mathtt{pc}) \mapsto \mathtt{short}\}\}$
$\mathtt{act3} : \mathtt{z} := \mathtt{z} \lhd \{\mathtt{pc} + 1 \mapsto \mathtt{z}(\mathtt{pc}) + 1\}$
$\mathtt{act4} : \mathtt{v} := \mathtt{v} \lhd \{\mathtt{pc} + 1 \mapsto \mathtt{v}(\mathtt{pc})\}$

end

Algorithm: Faulty Model

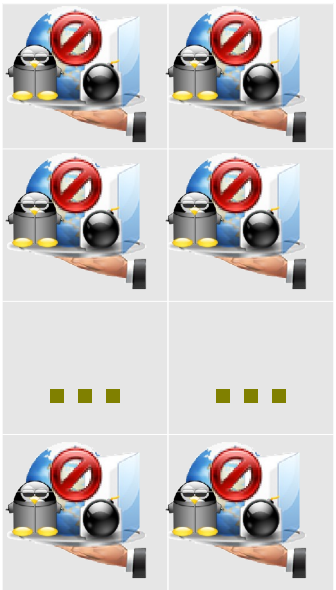
EVENTS
Event $evt\_sload\_11\_24\_EUT \mathrel{\widehat{=}}$
any
...
where
$grd : \mathtt{halt} = \mathtt{FALSE} \land \mathtt{pc} < \mathtt{maxpc}$
$grd\_t : \mathtt{z(pc)} > \mathtt{maxstack} - 1 \land \mathtt{index} \in 1 \mathinner{..} \mathtt{maxlocalvar} \land$
$\neg \mathtt{v(pc)(index)} = \mathtt{short}$
$grd\_EUT : \mathtt{eut} = \mathtt{FALSE}$
then
...
$act\_EUT : \mathtt{eut} := \mathtt{TRUE}$
end

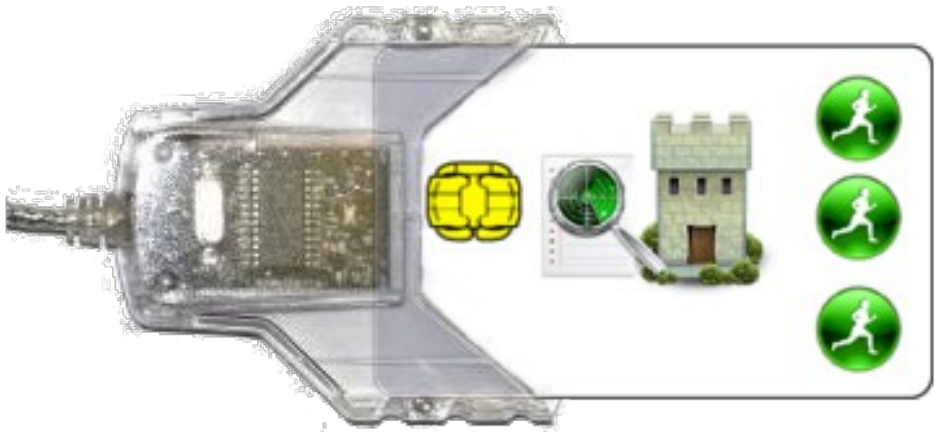
Test Generation Results

Depth	Time	Nb of tests	Speed ($sec/test$)	Model coverage	Manual plan coverage
3	1min30	30	3,0	13%	12%
4	12min30	432	1,7	33%	31%
5	1h30	10133	0,5	100%	93%
5 Full	2h30	10133	0,9	65%	93%
*	1h05	7318	0,5	100%	93%

Table : Abstract test generation evaluation

Smart Card Execution





- Execution of 5 cards
- Multiple failures discovered (exploitable or not)

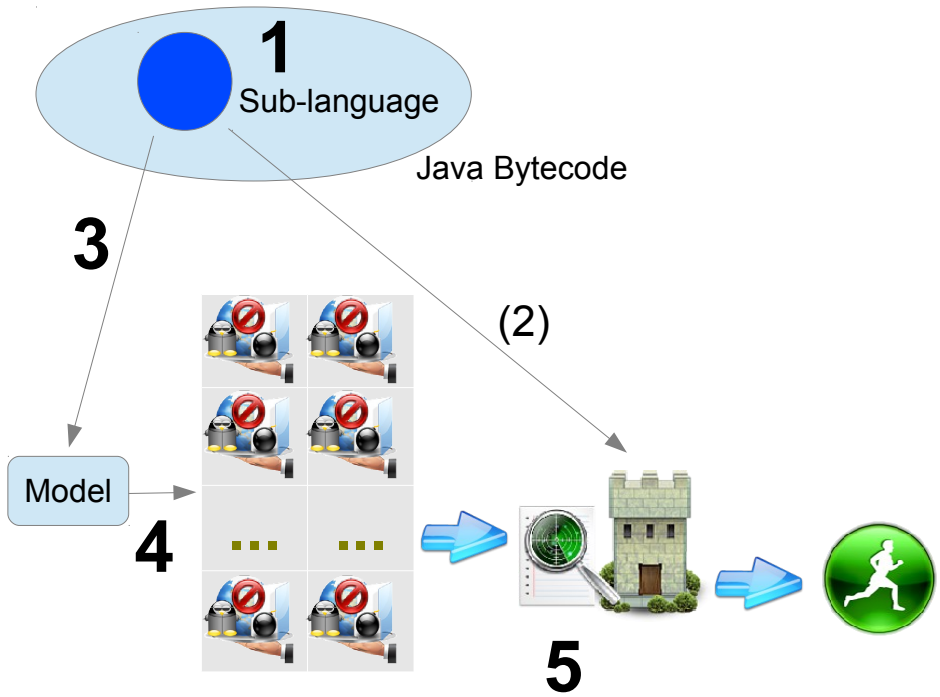
Conclusion and Future Works

Conclusion:

- Encouraging case study

Future Works:

- Test the entire Java Card language
- Deal with scaling problems
- Finish to implement tools



Contact

- Aymerick Savary :
  - www.aymericksavary.fr
  - aymerick.savary@etu.unilim.fr
  - aymerick.savary@usherbrooke.ca
- Jean-Louis Lanet :
  - jean-louis.lanet@unilim.fr
- Marc Frappier :
  - marc.frappier@usherbrooke.ca