iAL, the iTesla Action Language

01/06/17

What is the iAL?
Why was it developped?
Applications
Perspectives

Content

What is the iAL?

A Domain Specific Language for action simulation on a network

→ The « iAL » is a Domain Specific Language (DSL) written in Java and Groovy.

→ More flexible and scalable than existing software (Convergence).

→ Easier to use than generic langages (no need to be an IT specialist)

→ Can be interpreted by simulators thanks to a standard syntax.

DSL: computer language specialized to a particular application domain.

Concepts

Three concepts form the basis of the language:

- the contingency list
- the definition of actions
- the definition of rules include logical conditions and actions

Business rules engine

→ A software system to evaluate business rules and apply relevant actions

→ Handle a complex algorithm with micro rules that often change

→ Allow rules modification without the need for IT

Syntax - Contingencies

→ N-1 contingency

```
contingency(‘contingency-id’) {
    equipments ‘equipment-id‘
}
```

→ N-K contingency

```
contingency(‘contingency-id’) {
    equipments ‘equipment1-id‘, ‘equipement2-id’
}
```

Syntax - Contingencies

→ Automatic contingency lists

```
import eu.itesla_project.iidm.network.Country

for (l in network.lines) {
    s1 = l.terminal1.voltageLevel.substation
    s2 = l.terminal2.voltageLevel.substation

    if  (s1.country != s2.country) {
        contingency(l.id) {
            equipments l.id
        }
    }
}
```

Syntax - Actions

→ Pre-defined tasks

```
action(‘action-id’) {
    description "this is a description"
    tasks {
        openSwitch ‘switch1-id’
        closeSwitch ‘switch2-id’
        optimizePhaseShifterTap ‘pst-id’
    }
}
```

Syntax - Actions

→ Script tasks

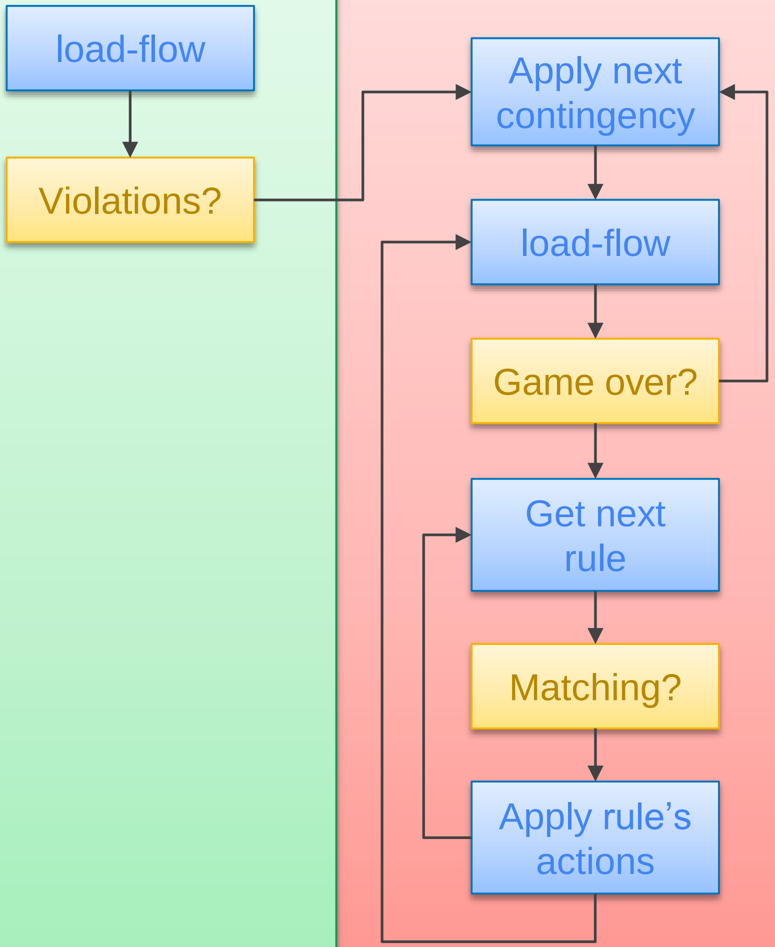
```
import static eu.itesla_project.iidm.network.PhaseTapChanger.RegulationMode.FIXED_TAP

action(‘action-id’) {
    description "this is a description"
    tasks {
        script {
            // switch is a reserved groovy keyword
            switch_(‘switch1-id’).open = true
            _switch(‘switch2-id’).open = false

            transformer(‘pst-id’).phaseTapChanger.regulationMode = FIXED_TAP
            transformer(‘pst-id’).phaseTapChanger.tapPosition = 25
        }
    }
}
```

Syntax - Rules

```
rule(‘rule-id’) {
    description "This is a description"
    when (condition1 || condition2) && condition3
    apply ‘action1-id’, ‘action2-id’
    life 1 // allow to use this rule only once
}
```

Pre-contingency state

Post-contingency state



load-flow

Violations?

Apply next contingency

load-flow

Game over?

Get next rule

Matching?

Apply rule’s actions

Game over:
- Still violations
- Still alive rules
- Not too many tries

Matching:
- When condition is true
- Still alive

Advanced definition of contingencies for security analysis

→ Use GroovyDslContingenciesProvider implementation to take advantage of automatic contingency lists:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <componentDefaultConfig>
        <ContingenciesProviderFactory>
             eu.itesla_project.action.dsl.GroovyDslContingenciesProviderFactory
        </ContingenciesProviderFactory>
    </componentDefaultConfig>

    <groovy-dsl-contingencies>
        <dsl-file>$HOME/.itesla/contingencies.groovy</dsl-file>
    </groovy-dsl-contingencies>
</config>
```

Why was it developped?

New needs at the end of the FP7 project

→ XML based syntax used in the FP7 to describe remedial actions for optimization and simulation needs

→ Optimization needs decreased, simulation needs increased

→ A new way to describe remedial actions was necessary

New needs at the end of the FP7 project

→ Some basis concepts were re-used (contingencies, contraints, actions) but fundamentally new possibilities appeared

→ Actions were limited to a few possibilities and binded to constraints and/or contingencies.

Now:

- An action can be any kind of change in the data model
- A logical condition can be applied to any data in the model
- Actions can be chained

Flexibility and scalability (why?)

→ Intermediate between black box tools (Convergence) and a generic programming language

→ Complex remedial actions can be implemented

→ Specific needs (optimize a tap position, model « la Durance ») can be added as plugins

Flexibility and scalability (how?)

→ Use the script task to describe a complex task

→ Create your own plugin to extend the Action DSL

Flexibility and scalability (how?)

→ PstTapPositionTaskExtension.groovy

```groovy
import com.google.auto.service.AutoService
import eu.itesla_project.action.dsl.spi.DslTaskExtension
import eu.itesla_project.contingency.tasks.ModificationTask

@AutoService(DslTaskExtension.class)
class PstTapPositionTaskExtension implements DslTaskExtension {
   @Override
   void addToSpec(MetaClass tasksSpecMetaClass, List<ModificationTask> tasks, Binding binding) {
       tasksSpecMetaClass.pstTapPosition = { String id, int tapPosition ->
           tasks.add(new PstTapPositionTask(id, tapPosition))
       }
   }
}
```

→ PstTapPositionTask.java

```java
package com.rte_france.itesla.action.util;

import eu.itesla_project.computation.ComputationManager;
import eu.itesla_project.contingency.tasks.ModificationTask;
import eu.itesla_project.iidm.network.*;
import java.util.Objects;

import static eu.itesla_project.iidm.network.PhaseTapChanger.RegulationMode.FIXED_TAP;

public class PstTapPositionTask implements ModificationTask {
    private final String phaseShifterId;
    private final int tapPosition;

    public PstTapPositionTask(String phaseShifterId, int tapPosition) {
        this.phaseShifterId = Objects.requireNonNull(phaseShifterId);
        this.tapPosition = tapPosition;
    }

    @Override
    public void modify(Network network, ComputationManager computationManager) {
        TwoWindingsTransformer phaseShifter = network.getTwoWindingsTransformer(phaseShifterId);
        phaseShifter.getPhaseTapChanger().setRegulationMode(FIXED_TAP);
        phaseShifter.getPhaseTapChanger().setTapPosition(tapPosition);
    }
}
```

→ Old action.groovy

```
import static eu.itesla_project.iidm.network.PhaseTapChanger.RegulationMode.FIXED_TAP

action(‘action-id’) {
    description "this is a description"
    tasks {
        script {
            transformer(‘pst-id’).phaseTapChanger.regulationMode = FIXED_TAP
            transformer(‘pst-id’).phaseTapChanger.tapPosition = 25
        }
    }
}
```

→ New action.groovy

```
action(‘action-id’) {
    description "this is a description"
    tasks {
        pstTapPosition(‘pst-id’, 25)
    }
}
```

Actions and uncertainties

→ Adressing the uncertainties is one of the key aspects of iTesla. The interaction between the description of remedial actions and this aspect must be taken into account.

→ Examples:

- Now: handling preventive remedial actions with logic rather than timestamps
- Tomorrow: applying different actions depending on the sample

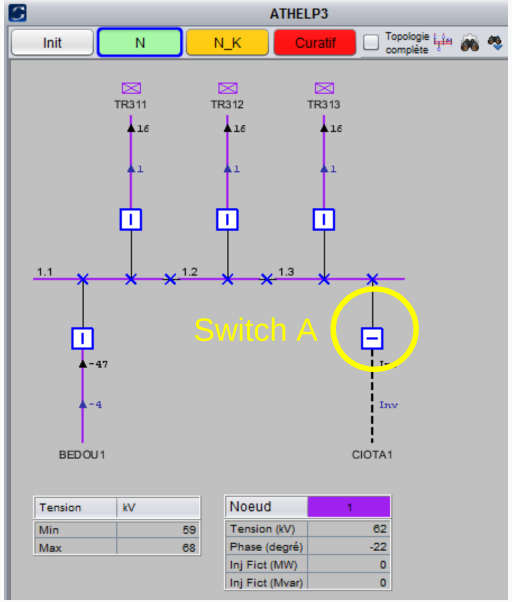
Examples

A simple preventive remedial action

→ Overloads can appear in pre-contingency state on several line an area around the Athelia 63 kV substation (Athelia – Ciotat 63 kV, Athelia – Bedoule 63 kV, Ciotat – Pont d’Aran – St Cyr 63 kV)

→ Rules implemented :

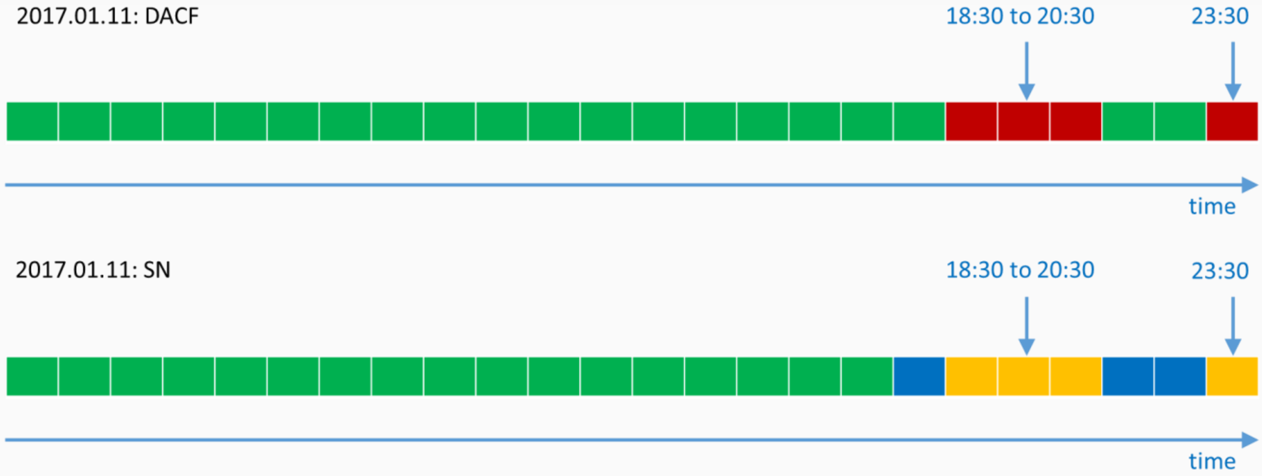
1. If switch A is closed AND there is at least one overload on the monitored lines, then open switch A.
2. If switch A is open AND was not just open because of the previous rule, then close switch A.



Tension	kV
Min	59
Max	68

Noeud	1
Tension (kV)	62
Phase (degré)	-22
Inj Fict (MW)	0
Inj Fict (Mvar)	0

A simple preventive remedial action



2017.01.11: DACF

18:30 to 20:30

23:30

time

2017.01.11: SN

18:30 to 20:30

23:30

time

- 🟩 Closed switch, no overload
- 🟥 Closed switch, needs opening
- 🟨 Open switch, closing creates overload
- 🟦 Open switch, closing is possible

A « manual automaton »

→ It is not possible to handle differently regulations in pre and post contingency state in Convergence which is a problem simulating the Camporosso PST. It can be done with the iAL.

→ Rules implemented:

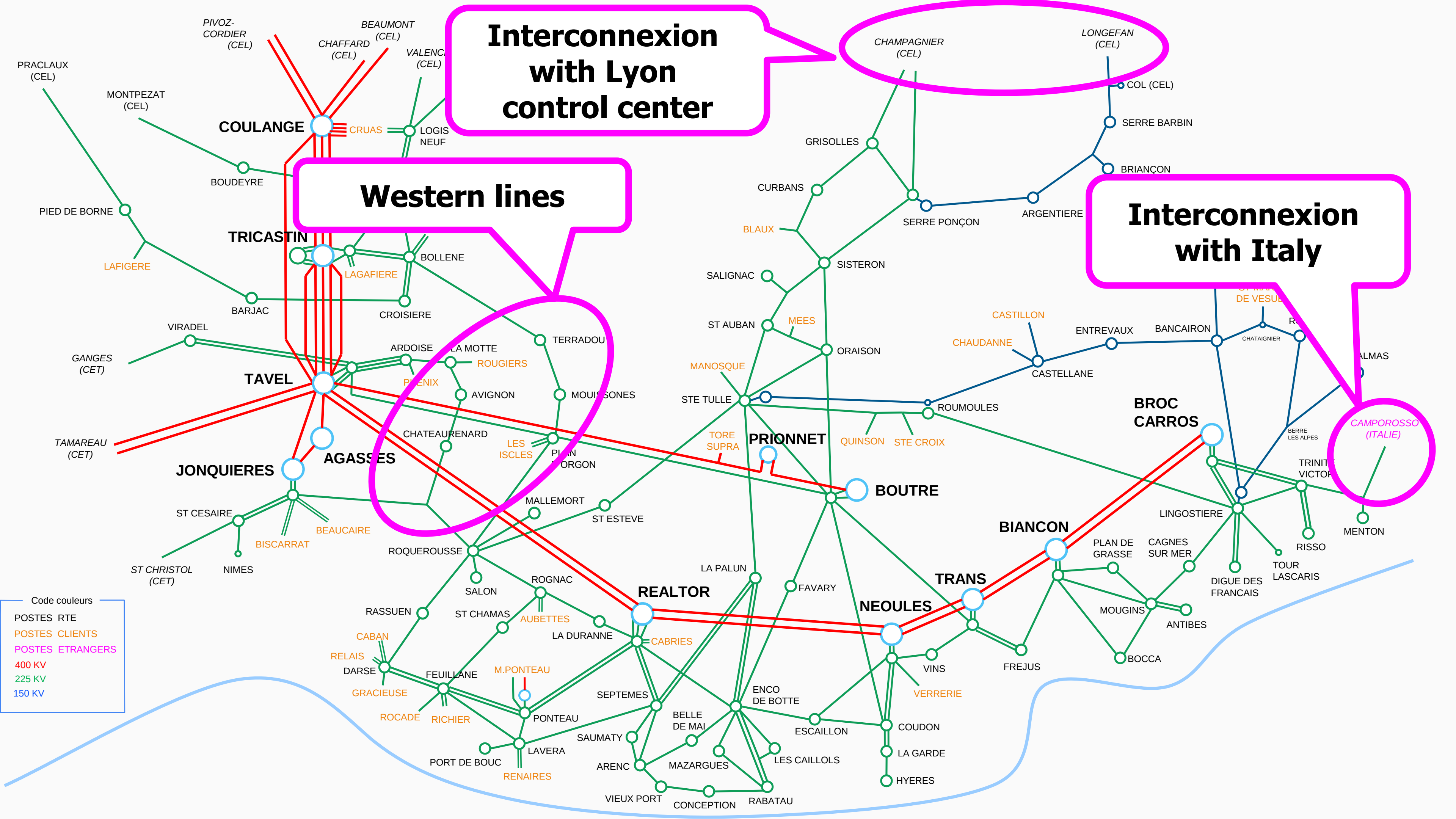
1. If the flow from France to Italy is above the setpoint value in pre-contingency state, decrease tap position until setpoint value ± dead band is reached.
2. If the flow from France to Italy is under the setpoint value in pre-contingency state, increase tap position until setpoint value ± dead band is reached.

A complex contingency: N-1 Tavel-Realtor 400 kV

After N-1 Tavel-Realtor 400 kV, severe overloads can imply successive remedial actions on varied elements of the network:

1. Switches
2. PST
3. Generation units

The choice of an action depends on the result of the previous action (load transfers).



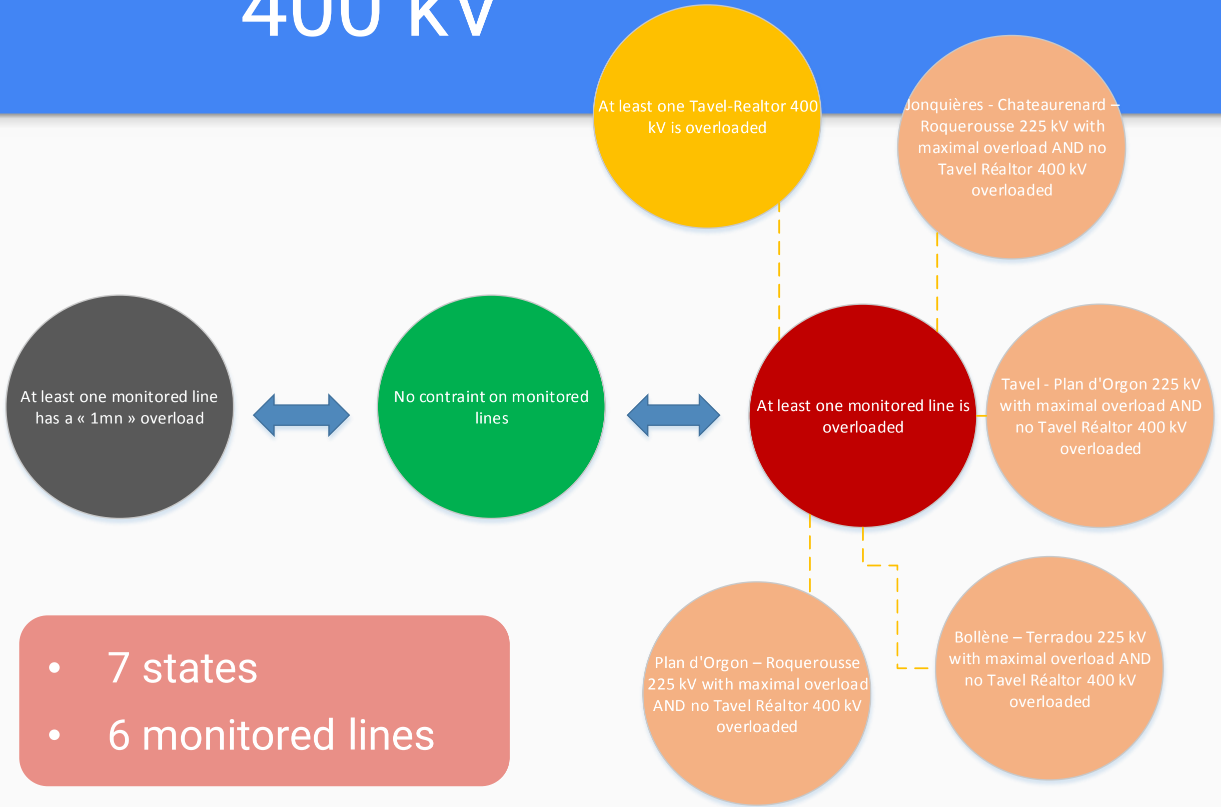
Interconnexion with Lyon control center

Western lines

Interconnexion with Italy

A complex contingency: N-1 Tavel-Realtor 400 kV

Monitored lines : Jonquières - Chateaurenard – Roquerousse, Bollène – Terradou, Plan d'Orgon – Roquerousse 225kV + Tavel - Plan d'Orgon, Tavel-Realtor 1 and 2 400kV



At least one monitored line has a « 1mn » overload

No contraint on monitored lines

At least one monitored line is overloaded

At least one Tavel-Realtor 400 kV is overloaded

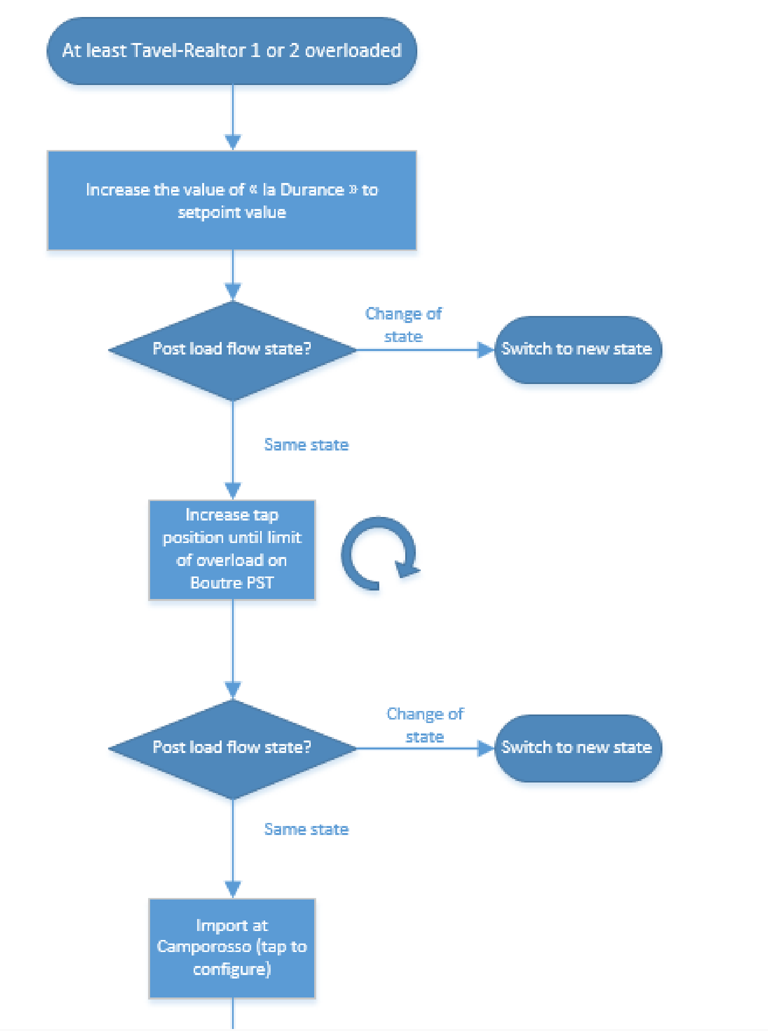
Jonquières - Chateaurenard – Roquerousse 225 kV with maximal overload AND no Tavel Réaltor 400 kV overloaded

Tavel - Plan d'Orgon 225 kV with maximal overload AND no Tavel Réaltor 400 kV overloaded

Plan d'Orgon – Roquerousse 225 kV with maximal overload AND no Tavel Réaltor 400 kV overloaded

Bollène – Terradou 225 kV with maximal overload AND no Tavel Réaltor 400 kV overloaded

- 7 states
- 6 monitored lines


At least Tavel-Realtor 1 or 2 overloaded
Increase the value of « la Durance » to setpoint value
Post load flow state?
Change of state
Switch to new state
Same state
Increase tap position until limit of overload on Boutre PST
Post load flow state?
Change of state
Switch to new state
Same state
Import at Camporosso (tap to configure)

→ The hypothesis described in our simulation tool was re-written in the iAL

→ « Optimization » functions can be added as plugins in the iAL

→ Some parameters can be configured

Perspectives

Interaction with action databases

New project to create a centralized database for remedial actions and automatons.

Goal: use the actions in automated security analysis processes.

Use cases:
⟶ Optimization usage: find the best set of actions for given cases to solve security issues
⟶ Simulation usage: check if the chosen actions actually solve the security issues

A format has to be specified. Optimization use case not yet supported by iAL.

Interaction with dynamic simulators

In the current version, the actions are successively computed with load flows.

The static approach is not always valid (voltage issues, tap changers, automatons…).

A version of the iAL compatible with dynamic simulators is currently under development.

The use of predefined tasks allows the translation to something understandable by a dynamic simulator (Modelica). Open discussion: what to do with script tasks?

IT developments

Functional improvements:

- Support busbar section contingencies
- Render the outputs
- Improve end user experience (error messages, user friendly editor…)

IT developments

Action Language improvements:

- Create plugins for common tasks
- Allow creation of plugins for conditions
- Increase code coverage up to 70%

Road-map: https://github.com/itesla/ipst-core/projects/2

Conclusion

To infinity, and beyond!

Open and flexible

Easy to use

Many usages of DSLs to explore