Formal methods for Safety Assessment of Critical Software at RATP

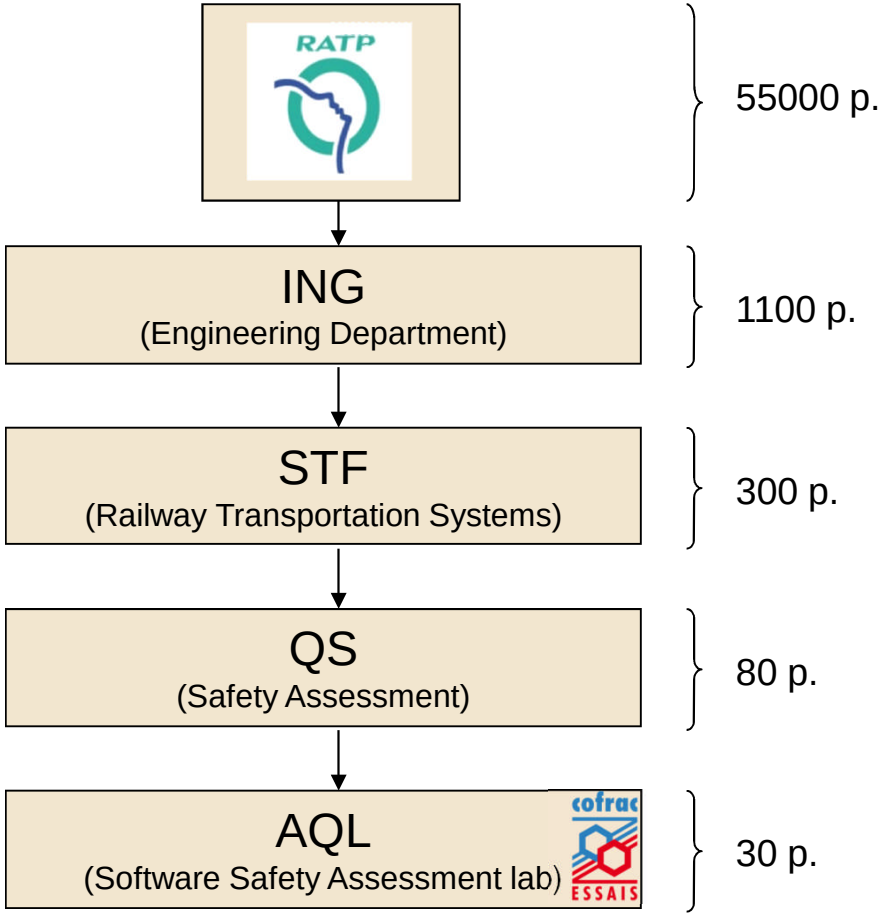
Engineering Department -- RATP/ING/STF/QS/AQL
Evguenia DMITRIEVA / Oct 16 2014, Toulouse

Plan

1. Industrial Context
2. Formal methods for software safety assessment : PERF
3. Use Cases
4. Conclusion

RATP's Software Safety Assessment lab

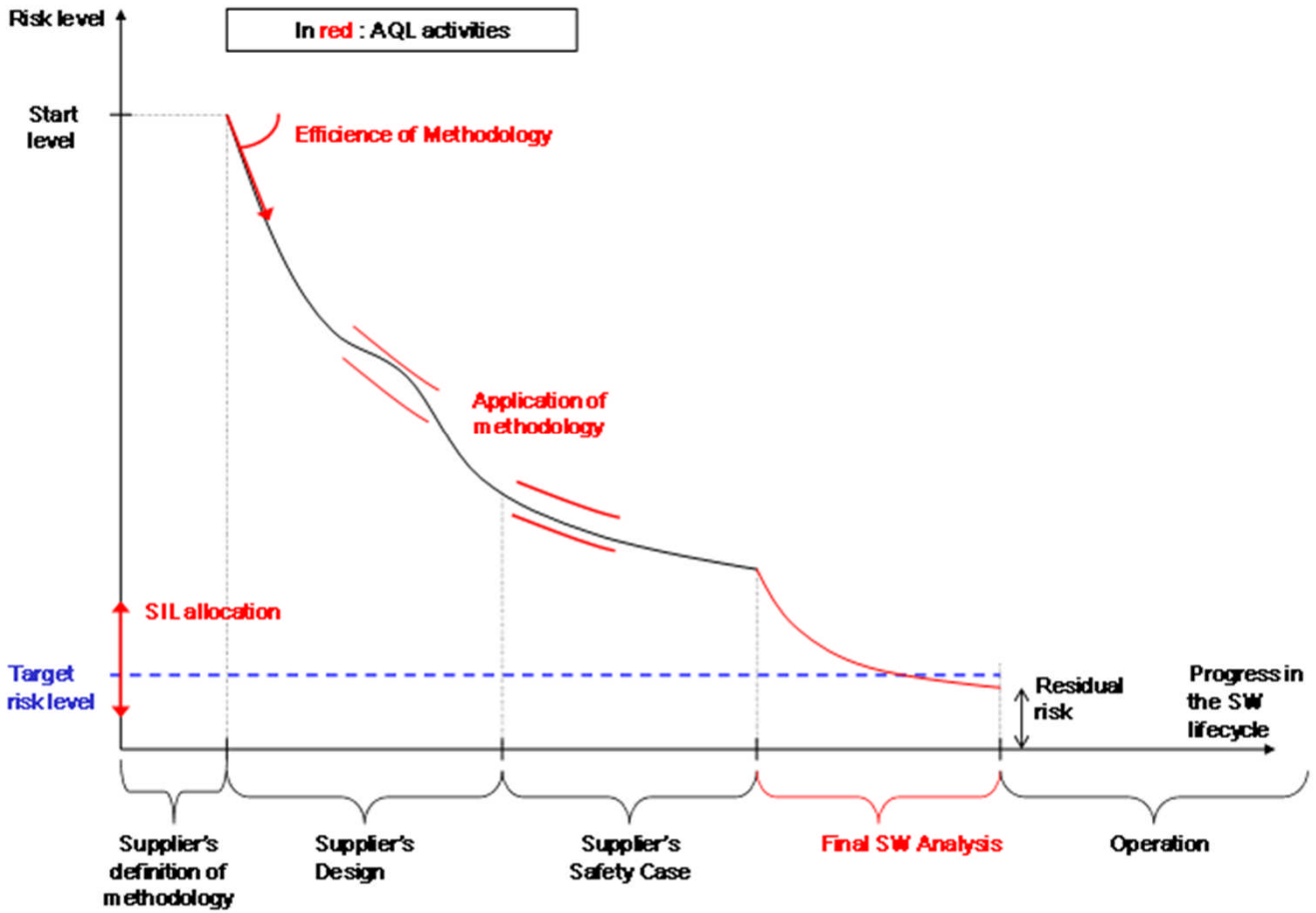
- Primary mission: internal assessment of safety critical software (ATP, Interlocking)
- Created in 1990: almost 25 years of experience
- AQL team: 30 persons, 50% engineers and PhD, 50% technicians
- Accredited by COFRAC, the French accreditation body
- Customers: internal (RATP) or external



AQL mission: independent assessment of safety critical software

- Check of safety cases provided by suppliers
- Additional analysis and verifications wherever supplier's methodology thought to be weak
- Covering the whole V lifecycle

AQL interventions



Why using Formal Proof ?

SACEM (pre-CBTC / 1989): retro-engineering (Z method) and formal proof

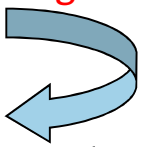
- 10 unsafe bugs found that had not been discovered with the « classical process » based on tests

METEOR (driverless CBTC / 1998): developed with B method

- get rid of of unit and integration tests
- delivery of a safe software at « first shot » (no need for other release after commissioning).

How using Formal Proof ?

- **90s:** (example: METEOR project)
  - RATP forces its supplier to use a Formal Method allowing Formal Proof (B method)
- **Since the 2000s:** (example: Computer Based Interlocking)
  - Public procurement laws: in a tender, it is forbidden to favor a supplier
  - Forcing the use of formal methods = a way to favor some suppliers
  - => RATP is only allowed to encourage the use of Formal Proof

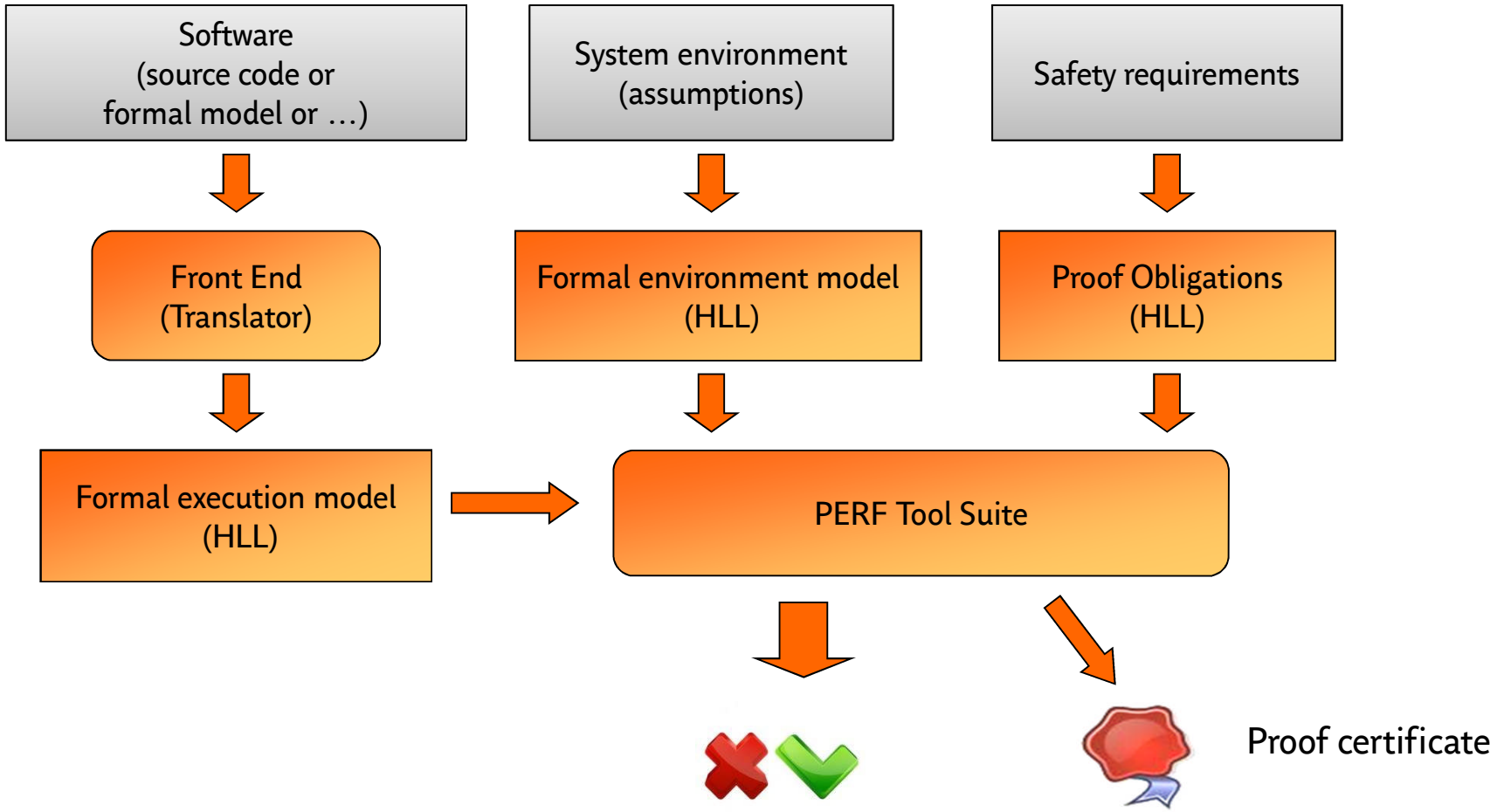


- RATP asked Prover Technology Company to develop a high integrity level ("SIL4") Formal Proof Tool Suite ("Prover Certifier")
- **Goal:** providing means to perform the formal verification, a posteriori, of a product that was not developed using a proof based process (like B method).

The PERF approach

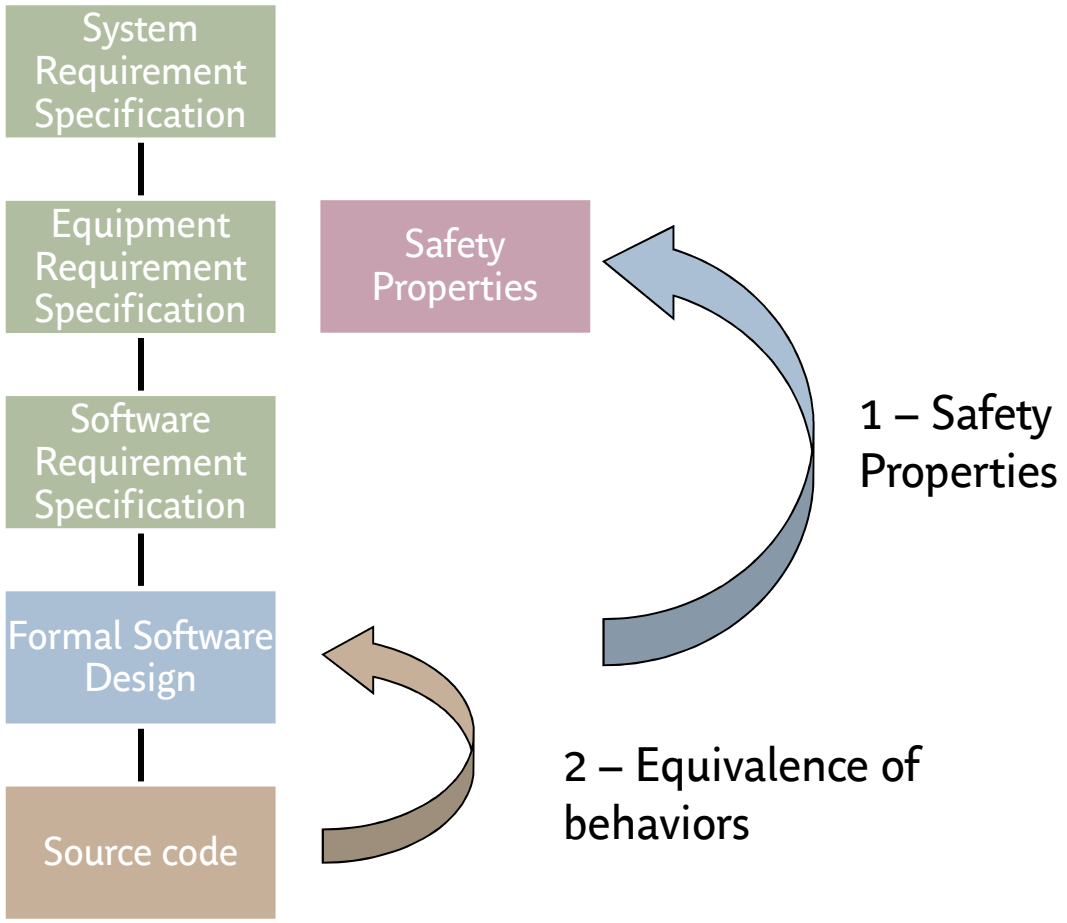
- **PERF** = **P**roof **E**xecuted over a **R**etroengineered **F**ormal model
- PERF = **P**reuve d'**E**valuation par **R**etromodélisation **F**ormelle
- Principle: using formal proof and techniques to verify properties over an **already developed** software product
- Techniques:
  - Basic synchronous modelling language (HLL)
  - Proof engine using SAT solving, k-induction, proof certification, …
- Scope:
  - Applicative SW (not low-level) + configuration data
  - Verification of « safety » properties (invariants)

The PERF approach



The PERF approach

- Formal verification of:
  - Safety Properties
  - Equivalence of behaviors



PERF Tool Suite

- Diversification, proof logging and proof checking
- CENELEC EN50128 compliant development process
- Independent assessment by RATP (AQL)

Use cases:

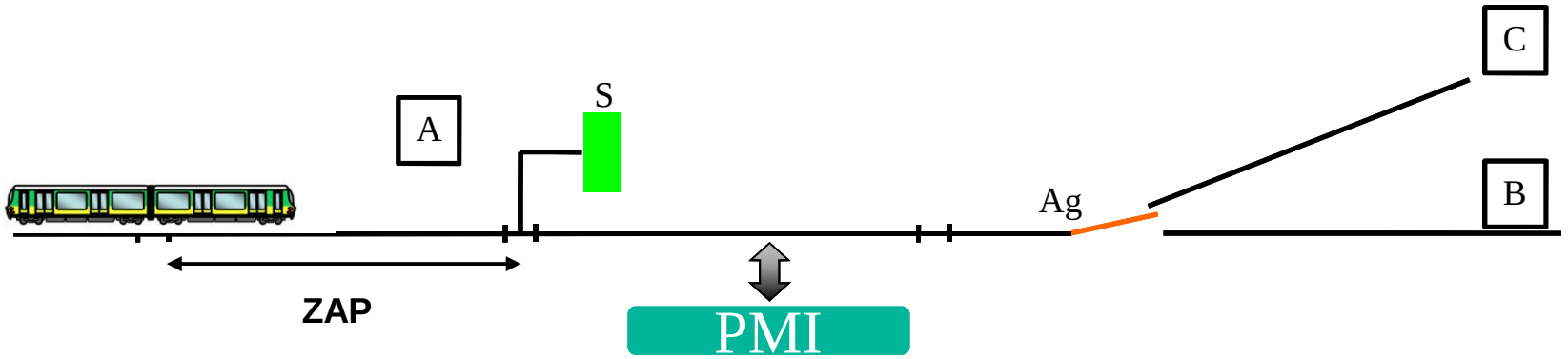
- *Verification of Safety Properties of SCADE models*
- *Verification of Safety Properties of C or Ada code*
- *Verification of equivalence beetween SCADE model and generated source code (C and Ada)*
- *Soon: Verification of Safety Properties of Relay Based Interlocking*

- The use of PERF has allowed to reveal and fix safety critical bugs (before commissioning) !

Computer Based Interlocking : PMI

Safety Hazards

- Derailment : on moving or badly set point, over speed
- Collision : front, rear, side
- Human operatives' Injuries : downgraded modes



Computer Based Interlocking : PMI

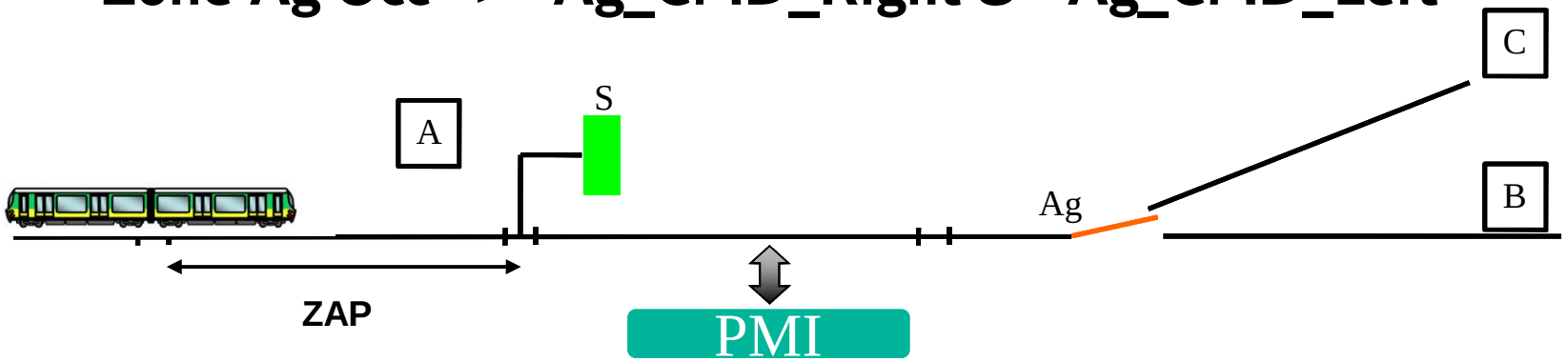
Safety Properties : Derailment on moving point

- *A point must not be controlled when the train is approaching the signal*

Zone-ZAP-Occ => ~Ag_CMD_Right & ~Ag_CMD_Left

- *A point must not be controlled when the train is located at this point*

Zone-Ag-Occ => ~Ag_CMD_Right & ~Ag_CMD_Left

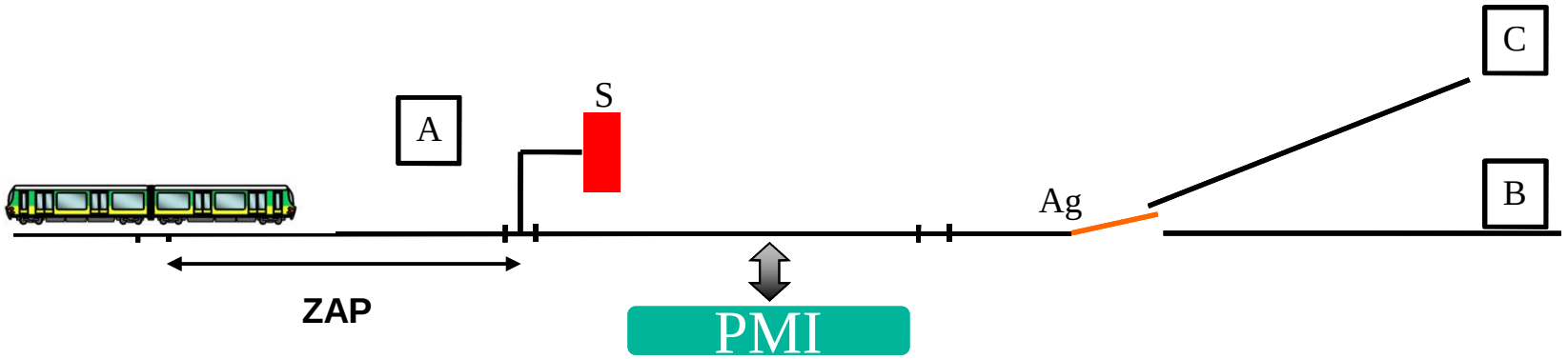


Computer Based Interlocking : PMI

Safety Properties : Derailment on moving point

- *A point must not be controlled when the train is approaching the signal*

Zone-ZAP-Occ => ~Ag_CMD_Right & ~Ag_CMD_Left



Computer Based Interlocking : PMI

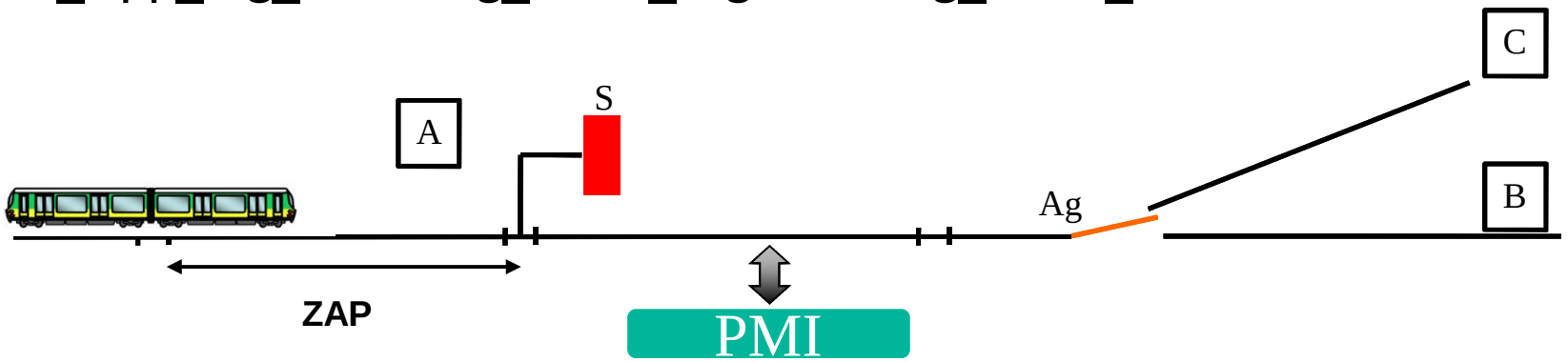
Safety Properties : Derailment on moving point

- *A point must not be controlled when the train is approaching the signal*

  ~~Zone-ZAP-Occ~~ => ~Ag_CMD_Right & ~Ag_CMD_Left

  Train_App_Sig_S := Zone_Zap_S_Occ & (Sig_S_G V Tempo_DA_S)

  Train_App_Sig_S => ~Ag_CMD_Right & ~Ag_CMD_Left



=> High level properties but the model of environment is rather complex

CBTC : Ouragan L13

Software Requirements (SEL)

- ⇒ software implements correctly its specification
- ⇒ often low-level and algorithmic
- ⇒ refinement system requirements into software one's should be verified otherwise
- ⇒ no need of environment model

Composant **SurveillerRecul_CC** Exigeance **SEL_Bord_SurveillerRecul_CC_0002**

Dans l'état « Opérationnel BORD », le composant SurveillerRecul_CC doit vérifier le domaine de définition de ses données d'entrée.

Dans le cas où une des entrées au moins dépasse son domaine admissible, le composant n'exécute pas ses traitements et génère une alarme « out of range ». Les sorties prennent les valeurs par défaut ci-dessous. Dans le cas contraire, le composant doit exécuter ses traitements.

Les valeurs par défaut des interfaces de sorties sont définies tel que

Interface	*Valeur par défaut*
PFU_SurveillerReculTrain_REC.IFU_ReculIntempestif	*C_DEMANDE_FU_DEFAUT*

==========

Formalisation de l'exigence en HLL

InRange(v) := v : [C_VITESSE_MIN_mmps, C_VITESSE_MAX_mmps];

Vitesses_OutOfRange (VitessesTrain**)** := ~(InRange(VitessesTrain.Max) &InRange(VitessesTrain.Inst) &
InRange(VitessesTrain.Min));

Prop_SEL_REC_02 := (Vitesses_OutOfRange(PE_OdometrieVitesse_ODO.VitessesTrainBord) #
Vitesses_OutOfRange(PE_OdometrieVitesse_ODO.VitessesTrainMessagerie)) =>

(~ PFU_SurveillerReculTrain_REC. IFU_ReculIntempestif.ValiditeDemandeFU &
~PFU_SurveillerReculTrain_REC. IFU_ReculIntempestif.NonDemandeFU &
PFU_SurveillerReculTrain_REC. IFU_ReculIntempestif.ContextePPFU = C_CONTEXTE_FU_DEFAUT)

CBTC : Ouragan L13

Equipment or System Requirements (DCSS)

- ⇒ Software implements correctly its system (equipment) requirements
- ⇒ Higher level of abstraction
- ⇒ Independent of implementation choices
- ⇒ No need to verify SRS
- ⇒ Environment model may be needed but not so complex

FT.BORD.3.1.2 - Elaborer le sens de marche requis du train **[DCSS-BORD-REQ-0156]**

Rôle :

Cette fonction détermine le sens de marche requis du train

Principes :

Le sens de marche requis est défini en fonction de la cabine active (cab A ou Cab B) et de la commande d'inversion du sens de marche dépendant du mode de conduite. Le sens de marche requis prend les valeurs Cabine A en tête ou Cabine B en tête comme défini dans le tableau suivant:

Cabine active	Inverser le sens de marche	Sens de marche requis
Cab A	Faux	Cab A en tête
Cab B	Faux	Cab B en tête
Cab B	Vrai	Cab A en tête
Cab A	Vrai	Cab B en tête
Indéterminée	*	Maintenu à sa dernière valeur
Aucune	*	Maintenu à sa dernière valeur

Le "sens de marche requis" est défini à "Cab A en tête" au démarrage du Bord.

Note: les * dans le tableau indiquent que la valeur ou l'état de la donnée peut prendre n'importe quelle valeur.

PO HLL

[DCSS-BORD-REQ-0156]

Types:

```
enum  {A, B, INDET, AUCUNE, CA_INVALID, CA_UNDEF} CabineActive_e;
enum  {AenTete, BenTete, SMR_INVALID} SensMarcheRequis_e;
enum  {Vrai, Faux, ISM_INVALID} InverserSensMarche_e;
```

Declarations:

```
//input "Inverser le sens de marche"
InverserSensMarche_e InverserSensMarche;
//input "Cabine active"
CabineActive_e CabineActive;
//output "Sens de marche requis"
SensMarcheRequis_e SensMarcheRequis;
(CabineActive_e * InverserSensMarche_e -> SensMarcheRequis_e) oracleSensMarcheRequis;
//valeur d'initialisation "Sens de marche requis" à l'init
SensMarcheRequis_e  C_SMR_INIT;
```

Definitions :

```
//pb de definition de la constante 'C_SENS_MARCHE_REQUIS_INIT‘, AenTete dans la spec et BenTete dans le code, non defini dans la spec de données statiques
C_SMR_INIT := BenTete;
```

PO HLL [DCSS-BORD-REQ-0156]

Definitions :

```
oracleSensMarcheRequis (cabAct, InvSM) := (cabAct, InvSM
                                | CA_INVALID , _ => pre<SensMarcheRequis_e>(SensMarcheRequis, C_SMR_INIT)
                                | _, ISM_INVALID => pre<SensMarcheRequis_e>(SensMarcheRequis, C_SMR_INIT)
                                | A , Vrai => BenTete
                                | B , Vrai => AenTete
                                | A , Faux => AenTete
                                | B , Faux => BenTete
                                | INDET, _ => pre<SensMarcheRequis_e>(SensMarcheRequis, C_SMR_INIT)
                                | AUCUNE,_ => pre<SensMarcheRequis_e>(SensMarcheRequis, C_SMR_INIT)
                                | CA_UNDEF, _ => SMR_INVALID
                        );
```

Proof Obligations :

```
//definition DCSS-BORD-REQ-0156 en tenant compte de 3 valuers elaborées par VOBC pour "Sens de marche requis"
InRange -> ( SensMarcheRequis = oracleSensMarcheRequis ( CabineActive, InverserSensMarche));
```

mapping HLL [DCSS-BORD-REQ-0156]

// mapping flux systeme et entrees/sorties des composants

Declarations:

```
(bool * bool * bool * bool -> CabineActive_e ) fCabineActive;
(bool * bool * bool -> SensMarcheRequis_e ) fSensMarcheRequis;
(bool * bool -> InverserSensMarche_e) fInverserSensMarche;
```

Definitions:

```
fInverserSensMarche (valid, value) := ( valid, value
                                        | false, _ => ISM_INVALID
                                        | true, true => Vrai
                                        | true, false => Faux);
InverserSensMarche := fInverserSensMarche (PM_Modes_MOD.'IM_InverserSensMarche'.'Validite',
PM_Modes_MOD.'IM_InverserSensMarche'.'DonneeBool');
```

mapping HLL [DCSS-BORD-REQ-0156]

```
fCabineActive (val, det, a, b) := (val, det, a, b
                                   | false, _, _, _=> CA_INVALID
                                   | true, true, true, false => A
                                   | true, true, false, true => B
                                   | true, false, false, false => INDET
                                   | true, true, false, false => AUCUNE
                                   | _, _, _, _=> CA_UNDEF);
CabineActive := fCabineActive (PE_CabineActiveModes_DPB.'IE_CabineActive'.'ValiditeCabineActive',
PE_CabineActiveModes_DPB.'IE_CabineActive'.'CabineActiveDetermine',
PE_CabineActiveModes_DPB.'IE_CabineActive'.'CabineAActive',
PE_CabineActiveModes_DPB.'IE_CabineActive'.'CabineBActive') ;

fSensMarcheRequis (valid, ba, indet) := (valid, ba, indet
                                         | false, true,true  => AenTete // pre (SensMarcheRequis,C_SMR_INIT)
                                         | false, false, true  => BenTete
                                         | true , true, _  => AenTete
                                         | true, false, _ => BenTete
                                         | _, _, _ => SMR_INVALID);

SensMarcheRequis := fSensMarcheRequis (PC_Sens_SEN.'IC_SensMarcheRequis'.'ValiditeSensMarcheRequis',
PC_Sens_SEN.'IC_SensMarcheRequis'.'SensMarcheRequisBA',
PC_Sens_SEN.'IC_SensMarcheRequis'.'SensMarcheRequisIndetermine');
```

ERTMS/ETCS level 2 : RFF/SNCF

Equipment requirements (DSL)

- Radio Block Center (RBC) : trackside equipment of ERTMS-level 2
  - Interface with local Interlockings (signal aspects, track occupations and route status, …)
  - Management of movement authorities for all trains within the controlled area
  - Management of trackside data
- Ada manually coded SW (~150 000 lines of code)

ERTMS/ETCS level 2 : RFF/SNCF

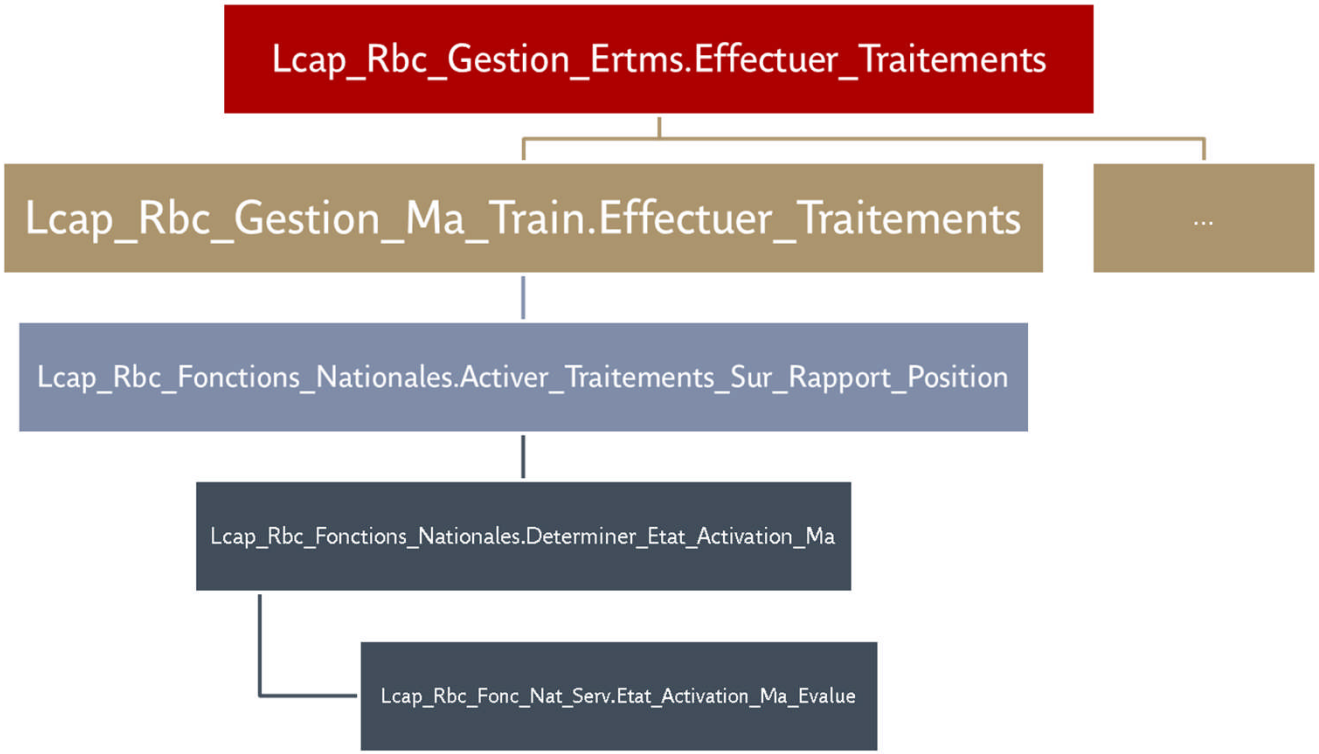
« *Au passage des modes OS, SR, SB ou PT vers le mode FS, ETAT_VL doit être égal à VOIE_LIBRE* »

Formalisation de l'exigence en HLL

```
(((
        ~(
                (('lcap_rbc_gest_donnee_train_ctxt.g_tab_deplacement_train'('du_train')).'infos_train'.'m_mode' ==
                                                        'lcap_types_variables_ertms.c_m_mode_stm_national')
                #
                (('lcap_rbc_gest_donnee_train_ctxt.g_tab_deplacement_train'('du_train')).'infos_train'.'m_mode' ==
                                                        'lcap_types_variables_ertms.c_m_mode_full_supervision')
                )
  )
  &
  ('etat_d_activation_du_ma'='lcap_types_donnees_train.c_req_ma_fs_nominal')
 )
 ->
                ('lcap_rbc_gest_donnee_train_ctxt.g_tab_etat_train'('du_train')).'etat_voie_libre' == 2
)
```

ERTMS/ETCS level 2 : RFF/SNCF

- Abstraction
- Concretisation
- Characterisation



Use of formal proof by RATP's suppliers

Line (Paris Metro)	System	Development Method	Formal Proof toolkit	Date of Operation	Usage
14	Driverless CBTC	B Method	Atelier B	1998	Safety Case
3	CBTC (Zone Controller)	B Method	Atelier B	2010	Safety Case
1	Driverless CBTC	B Method	Atelier B	2011	Safety Case
5	CBTC (Zone Controller)	B Method	Atelier B	2013	Safety Case
3	CBTC (Zone Controller)	Scade 5	Prover Certifier	2010	Safety Case
1	Compute Based Interlocking	Petri nets based graphs	Prover Certifier	2011	Safety Case
8	Computer Based Interlocking	Petri nets based graphs	Prover Certifier	2011	Safety Case
12	Computer Based Interlocking	Petri nets based graphs	Prover Certifier	2010	Safety Case

Use by RATP/AQL for Paris Metro

Line (Paris Metro)	System	Development Method	Formal Proof toolkit	Date of Operation	Usage
8 (2nd phase)	Computer Based Interlocking	Petri nets based graphs	Prover Certifier	2011	Safety Case
12	Computer Based Interlocking	Petri nets based graphs	Prover Certifier	2012	Safety Case
4	Computer Based Interlocking	Petri nets based graphs	Prover Certifier	2013	Safety Case
1 (3rd phase)	Computer Based Interlocking	Petri nets based graphs	Prover Certifier	2013	Safety Case
5	CBTC (Carborne Controller)	Scade 5	Prover Certifier	2013	Safety Assessment
13	CBTC	Scade 6	Prover Certifier	2014	Safety Assessment

RATP/AQL external missions

Storstockholms Lokaltrafik

- ISA

Transport Authority of a great metropolis (USA - *confidential*)

- ISA
- Formal Verification of Solid State Interlocking systems

Still ongoing

RFF (France – ERTMS project)

- Expert Opinion
- Formal Verification of safety properties of an Ada manually coded SW (~150 000 lines of code)

Why using PERF ?

- In an assessment process : PERF can take place concurrently with software test phases (reduction of projects global duration)
- Makes regression analysis very efficient and very quick (“push button”)
- Shows unexpected unsafe scenarios
- Leads to make explicit the assumptions (made to achieve the safety demonstration)


RATP