CLEARSY
SYSTEM ENGINEERING

La preuve formelle dans le développement logiciel embarqué

ESME SUDRIA CAP'TRONIC

23 juin 2015

Paris
Lyon
Aix

WWW.CLEARSY.COM

AUTEUR :
Etienne PRUN
Etienne.prun@clearsy.com

ClearSy
Spécialiste des systèmes sécuritaires

- Effectif : 100 (2014)
- CA : 7,3 M (2013)
- Métier
  - AtelierB : (R&D, Développement de l'outil)
  - Modélisation B (logiciel, système, Datas)
  - Réalisation de Systèmes sûrs (matériel, installation…)

Sommaire

- Partie 1 :
  - Principales Normes
  - Pourquoi les méthodes formelles ?
  - Principales techniques de vérification
- Partie 2 :
  - Application des Méthodes Formelles
  - Application de la méthode B

Normes IEC61508/EN50128

Logiciels et systèmes critiques :

Préservation de la vie humaine

- Les méthodes formelles « Highly recommended » par les normes industrielles pour le développement de systèmes de sécurité de niveau 4 (IEC 61508, EN50128)

Safety Integrity Level	Probability of a failure to danger per hour (PFH_D)
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

Normes DO-178 C

- 5 niveaux de criticité (Development Assurance Level) :

  Niveau A - un problème catastrophique - Sécurité du vol ou atterrissage compromis - Crash de l'avion

  Niveau B - un problème majeur entraînant des dégâts sérieux, voire la mort de quelques occupants

  Niveau C - un problème sérieux entraînant un dysfonctionnement des équipements vitaux de l'appareil

  Niveau D - un problème pouvant perturber la sécurité du vol

  Niveau E - un problème sans effet sur la sécurité du vol

- 2 paragraphes dans le DO178B à une annexe entière, le DO-333 «Formal Methods Supplement to DO-178C and DO-278A»

Normes

Sécurité des logiciels et systèmes :

Préservation de la confidentialité des données

EAL1 - functionally tested
EAL2 - structurally tested
EAL3 - methodically tested and checked
EAL4 - methodically designed, tested, and reviewed
EAL5 - semiformally designed and tested
EAL6 - semiformally verified design and tested
EAL7 - formally verified design and tested

EAL: Evaluation Assurance Level

Pourquoi les méthodes formelles ?

- Continuer à maîtriser les systèmes que l'on conçoit et construit
  - 18 M loc embarquées sur une voiture (2001)
  - ~50 fonctions réparties sur une centaine de calculateurs

Pourquoi les méthodes formelles ?

- L'allumage d'un téléphone basique (2005) :
  - exécute un code de ~100 000 lignes
  - représentant un automate à 7 niveaux et 700 états.
- Le séquencement du vol d'Ariane 5 (1996):
  - 95 tâches fonctionnant en parallèle.
- La spécification et la conception des automatismes d'une centrale nucléaire :
  - occupe 300 m linéaires de documentation papier.

Pourquoi les méthodes formelles ?

- La commande de vol de l'Airbus A380 comprend 1 million de lignes de code (2007)
  - contre 100 000 sur les premiers A320 (1987)

Pourquoi les méthodes formelles ?

On sait calculer la fiabilité d'une résistance

Modèle général associé à la famille

$$\lambda = \lambda_{Physique} \times \Pi_{PM} \times \Pi_{Process} \quad \text{avec :}$$

$$\lambda_{Physique} = \lambda_{0\_Résistance} \times \sum_{i}^{Phases} \left(\frac{t_{annuel}}{8760} \right)_i \times \left(\Pi_{Thermo-électrique} + \Pi_{TCy} + \Pi_{Mécanique} + \Pi_{RH} \right)_i \times \left(\Pi_{Induit} \right)_i$$

Facteur $C_{sensibilité}$

	Sensibilité relative (note sur 10)			
	EOS	MOS	TOS	$C_{sensibilité}$
Résistance fixe à couche à faible dissipation haute stabilité (RS), d'emploi courant (RC), « Minimelf »	5	2	4	3,85
Résistance fixe à couche à forte dissipation	2	3	1	2,25
Résistance fixe bobinée de précision à faible dissipation	2	1	3	1,75
Résistance fixe bobinée à forte dissipation	2	3	1	2,25
Potentiomètre non bobiné (CERMET)	1	5	2	2,50
Chip résistif	5	4	6	4,75
Réseau résistif CMS	4	5	3	4,25
Résistance de précision, haute stabilité à feuille métallique gravée	6	6	4	5,8

Mais pas celle d'un logiciel

- recours massif aux tests fonctionnels/d'intégration/unitaires
- qui peuvent être évités/réduits si recours aux méthodes formelles

Méthodes formelles

- Sont des techniques permettant de raisonner rigoureusement
- Avec l'aide de logique mathématique
- Sur des systèmes
- Pour démontrer leur validité par rapport à une certaine spécification.

Techniques de vérification

- Analyse statique par interprétation abstraite
- Model-checking
- Preuve de théorèmes

Analyse statique par interprétation abstraite

- Calcule symboliquement un sur-ensemble des états accessibles du système
- Parcourt le programme analysé en y remplaçant les valeurs par des abstractions.

Par exemple, les valeurs des variables entières sont abstraites par des intervalles d'entiers, ou des relations algébriques entre variables

- Traite principalement la correction du codage
  - **Division par zéro,**
  - **Non dépassement des bornes d'un tableau,**
  - **Erreur d'arrondi (flottant)**
  - **etc.**

Model-checking

- Analyse exhaustivement l'évolution du système lors de ses exécutions possibles
- Composition de modèles pour vérifier une propriété
- Mur de la complexité
  - r : A * A — 24*4 = 65536 possibilités pour r
  - r : P(A * A) — 224*4 = 265536 si card(A)=4

Preuve de théorèmes

- Démontrer des théorèmes sur les formules logiques décrivant la sémantique du programme

- Problème indécidable:
  - pas d'algorithme qui permette de conclure, pour la logique du premier ordre

Utilisation des Méthodes Formelles

A quel moment utiliser des MF ?

Dès que vous voulez garantir un raisonnement

- Modéliser formellement un système ou un contexte
- Prouver une propriété sur ce modèle

A quel moment utiliser des MF ?

Une analyse avec méthode formelle peut fournir la preuve que le système est complet et correct vis à vis de ses exigences.
L'utilisation de spécifications formelles seules rend les exigences non ambiguës.

Extrait de la norme RTCA DO-178B/EUROCAE ED-12B

A quel moment utiliser des MF ?

Pour les spécifications, des méthodes formelles mathématiques sont recommandées car le modèle formel fournit précision, non ambiguïté et cohérence.

Norme du ferroviaire

A quel moment utiliser des MF ?

L'utilisation de méthode formelle a pour but d'éviter et d'éliminer les erreurs de spécification, de conception et de codage lors du développement du logiciel.

Extrait de la norme RTCA DO-178B/EUROCAE ED-12B

Choisir de faire une preuve formelle

- Besoin d'une garantie de sécurité globale
- Besoin d'avoir exprimé toutes les conditions nécessaires à la sécurité et le "pourquoi c'est sécuritaire".

On sait que c'est assez rigoureux pour être formalisé seulement si on l'a formalisé

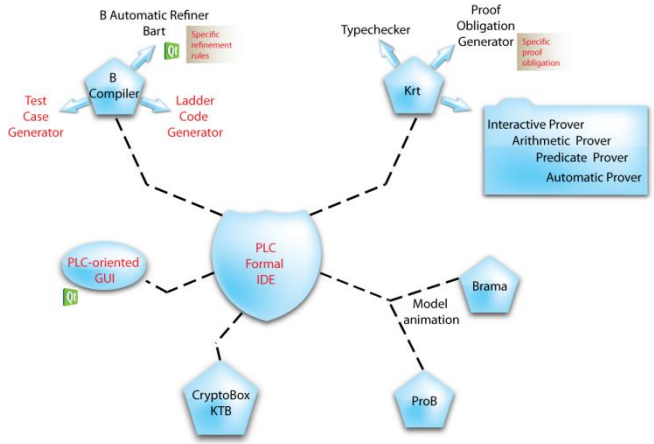
- La résistance d'une chaîne est celle du maillon le plus faible

Utiliser les méthode formelles pour renforcer un maillon ou pour démontrer qu'il n'y a pas de maillon faible

AtelierB

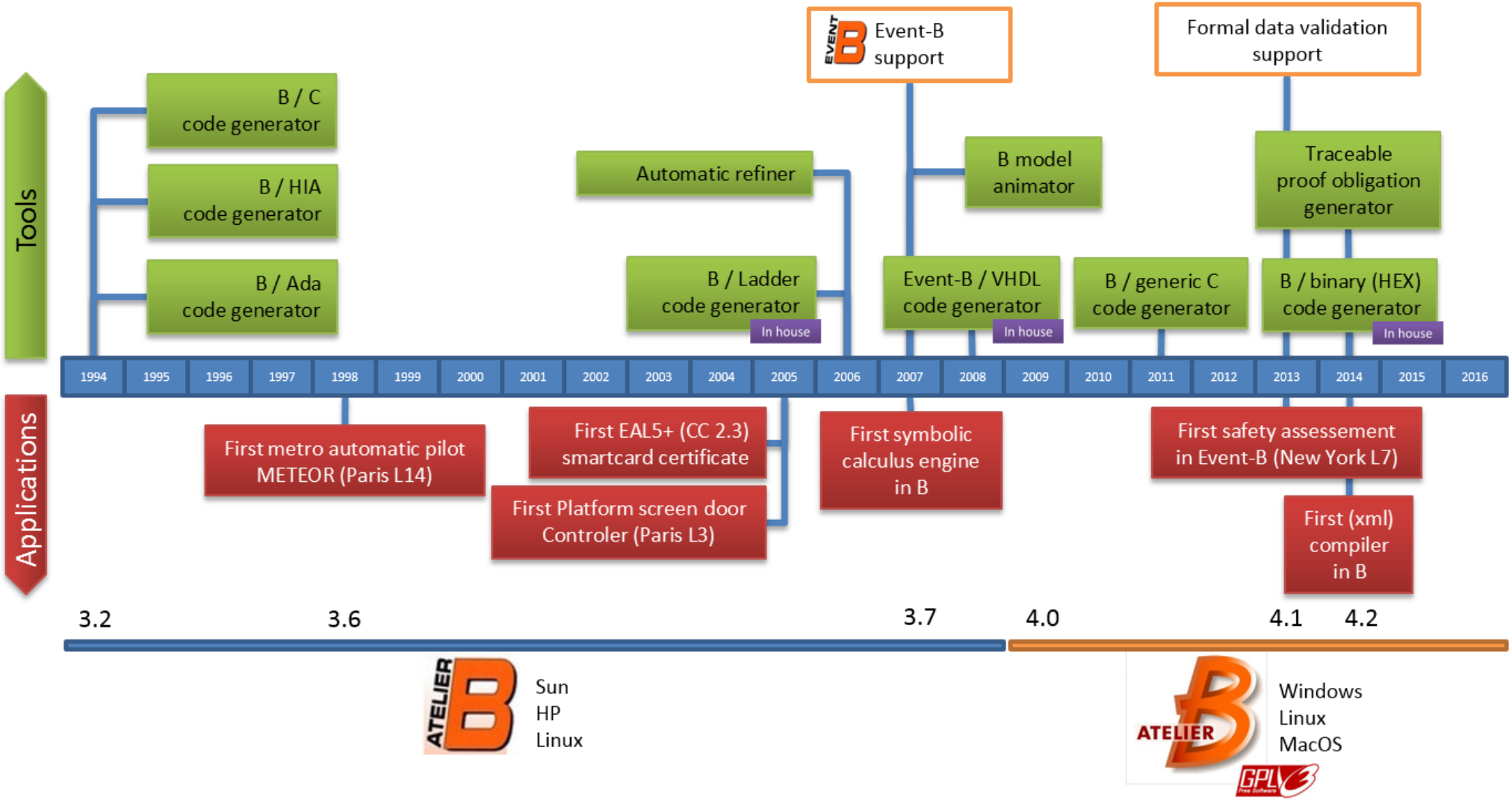
AtelierB

Atelier logiciel qui permet une utilisation opérationnelle de la méthode formelle B.



- Une aide à la preuve :
  - pour démontrer les obligations de preuve, grâce à des outils de preuve
- Une aide au développement :
  - Gestion des dépendances entre composants B, Le raffinement automatique, Analyse statique
- **Des outils de confort pour l'utilisateur :**
  - Représentation graphique de projets, Affichage de l'état d'un projet, Gestion des règles

AtelierB

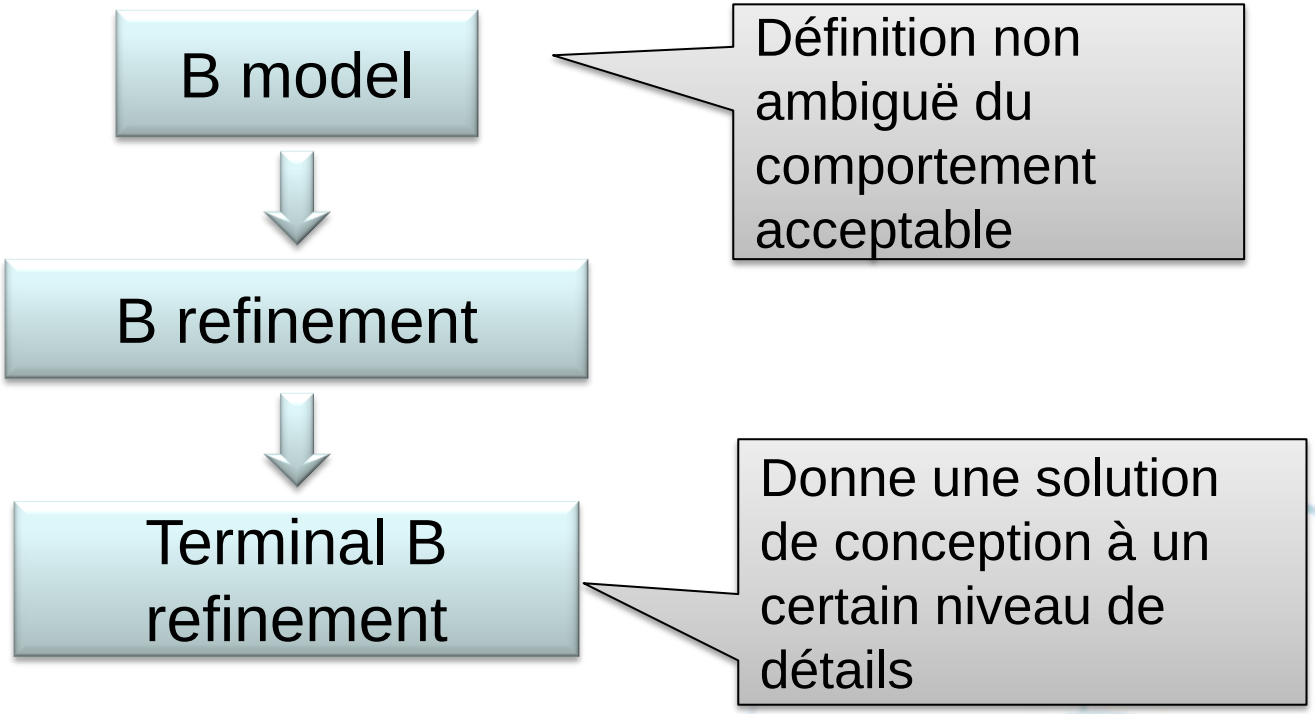

Tools
Applications
B / C code generator
B / HIA code generator
B / Ada code generator
Automatic refiner
B / Ladder code generator
In house
Event-B support
B model animator
Event-B / VHDL code generator
In house
B / generic C code generator
Formal data validation support
Traceable proof obligation generator
B / binary (HEX) code generator
In house
1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016
First metro automatic pilot METEOR (Paris L14)
First EAL5+ (CC 2.3) smartcard certificate
First Platform screen door Controler (Paris L3)
First symbolic calculus engine in B
First safety assessement in Event-B (New York L7)
First (xml) compiler in B
3.2
3.6
3.7
4.0
4.1
4.2
Sun
HP
Linux
Windows
Linux
MacOS
GPL

B événementiel / B système

B événementiel / B système

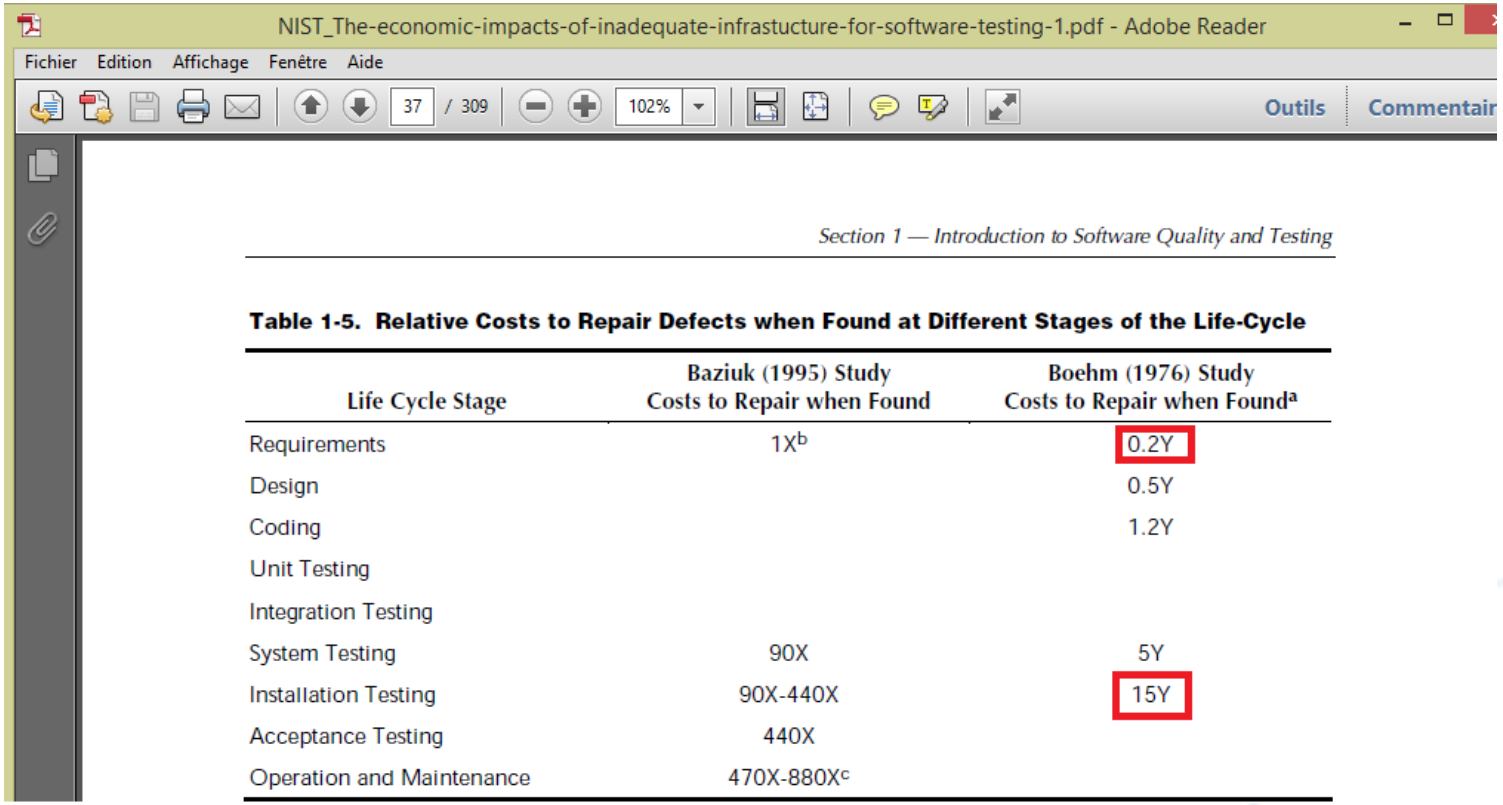
Utilisé pour décrire formellement les systèmes et raisonner mathématiquement sur leurs propriétés.

- Basé sur la Méthode B :
  - Formalisme
  - Preuve
  - Raffinement



Pourquoi des MF en spécification

- Le coût d'une détection d'anomalie en installation 75 fois plus important que celui en spécification :



Section 1 — Introduction to Software Quality and Testing

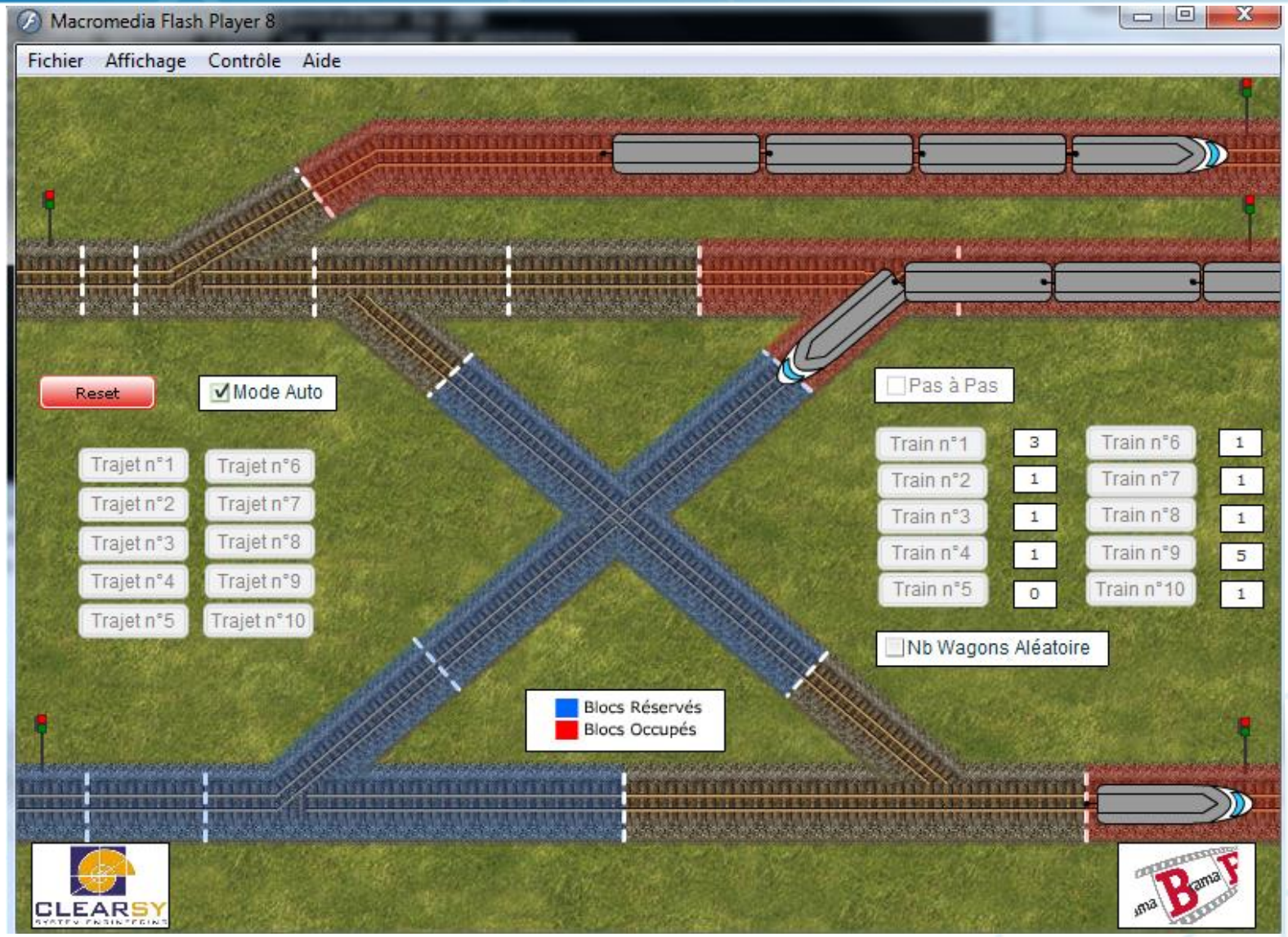
Table 1-5. Relative Costs to Repair Defects when Found at Different Stages of the Life-Cycle

Life Cycle Stage	Baziuk (1995) Study Costs to Repair when Found	Boehm (1976) Study Costs to Repair when Found[a]
Requirements	1X[b]	0.2Y
Design		0.5Y
Coding		1.2Y
Unit Testing		
Integration Testing		
System Testing	90X	5Y
Installation Testing	90X-440X	15Y
Acceptance Testing	440X	
Operation and Maintenance	470X-880X[c]	

Pourquoi des MF en spécification

- Fonctionnalités du système
  - Identifier clairement les fonctionnalités
  - Exprimer toutes les fonctionnalités du système
  - Garantir que les sous-fonctions font bien la fonctionnalité
- Hypothèses du raisonnement de sûreté
  - Identifier clairement
  - Les allouer correctement
  - Les exprimer toutes

Animation route



Exercice 05 (voies)

PROPERTIES

rtbl:BLOCS<->ROUTES &
dom(rtbl)=BLOCS &
ran(rtbl)=ROUTES &
nxt:ROUTES-->(BLOCS>+>BLOCS) &
fst:ROUTES-->BLOCS &
lst:ROUTES-->BLOCS &

Tous les blocs sont utilisés
Toutes les routes sont utilisées
Un bloc n'est le successeur que d'un seul bloc
Il n'y a qu'un bloc d'entrée d'une route
Il n'y a qu'un bloc de sortie d'une route

PROPERTIES :

!rr.(rr:ROUTES => nxt(rr):rtbl~[{rr}]-{lst(rr)}>->>rtbl~[{rr}]-{fst(rr)}) **?**

« Pour toute route, la fonction qui associe à un bloc de la route son successeur est une bijection entre les blocs non finaux de la route et les blocs non initiaux de la route »

Une route est un ensemble connexe de blocs

Exercice 05 (voies)

PROPERTIES

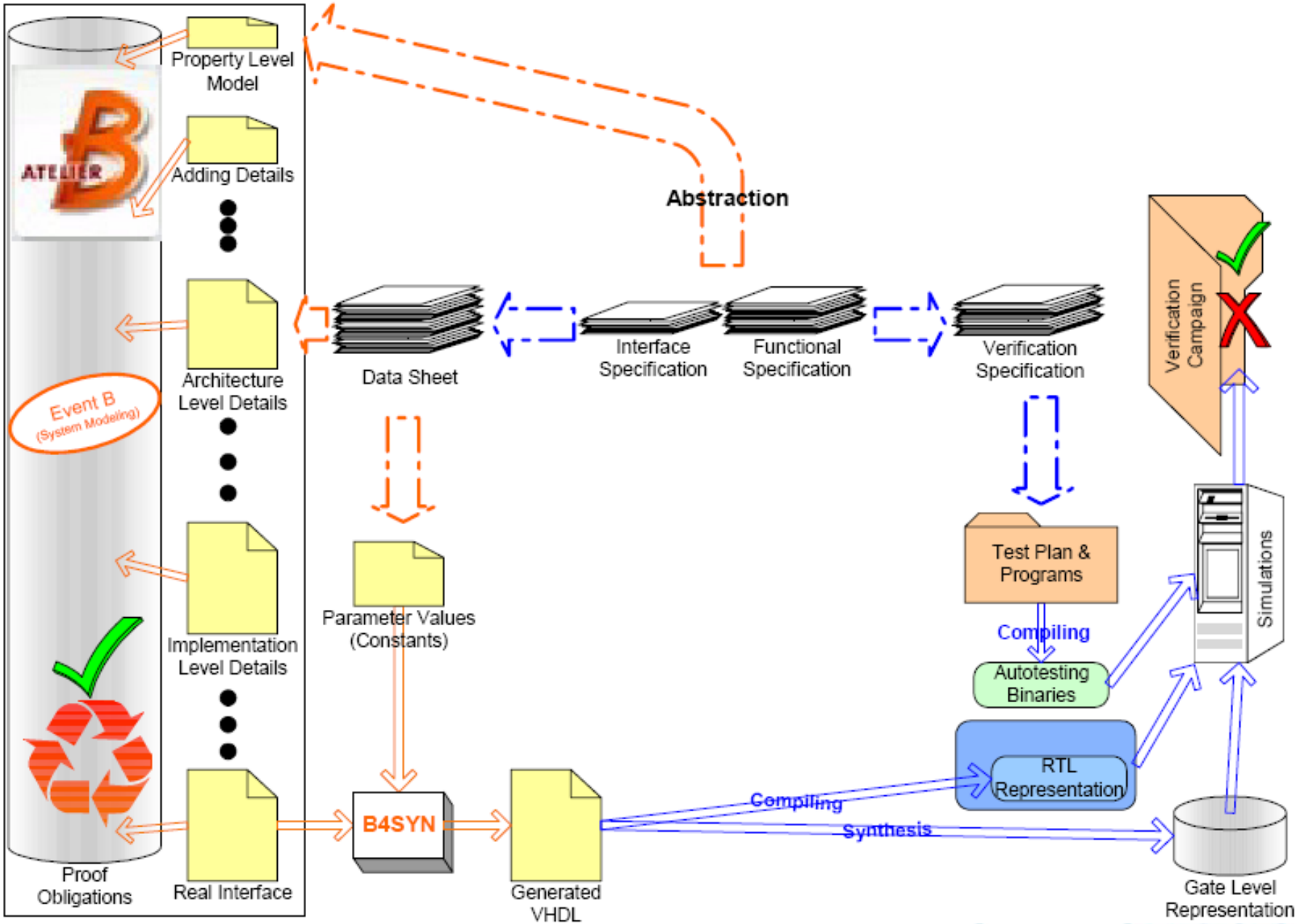
rtbl:BLOCS<->ROUTES &
dom(rtbl)=BLOCS &
ran(rtbl)=ROUTES &
nxt:ROUTES-->(BLOCS>+>BLOCS) &
fst:ROUTES-->BLOCS &
lst:ROUTES-->BLOCS &

Tous les blocs sont utilisés
Toutes les routes sont utilisées
Un bloc n'est le successeur que d'un seul bloc
il n'y a qu'un bloc d'entrée d'une route
Il n'y a qu'un bloc de sortie d'une route

PROPERTIES :
!rr.(rr:ROUTES => !SS.(SS<:BLOCS & SS<:nxt(rr)[SS] =>
SS={})) **?**

« Un bloc ne peut pas être son successeur »
Une route ne contient pas de cycle

New Proved Source HDL Development Flow



Trying a rough comparison

Criteria	Classical Flow	Experimental Flow
Effort *person.days*	~145	~145 (~200 translator)
Volume *commented source lines*	~3 600 VHDL	~7 500 B ~3 500 VHDL
Proof *number of items*	-	~1 600 obligations ~ 600 with ~4000 cmds
Structure *module number*	~15 tightly linked	4 loosely linked (only 1 generated module)
Simulation *RTL & gate level*	~30 patterns Ok	~30 patterns Ok (better debug signals)
Size *equivalent nand gates*	~5 Kgates	

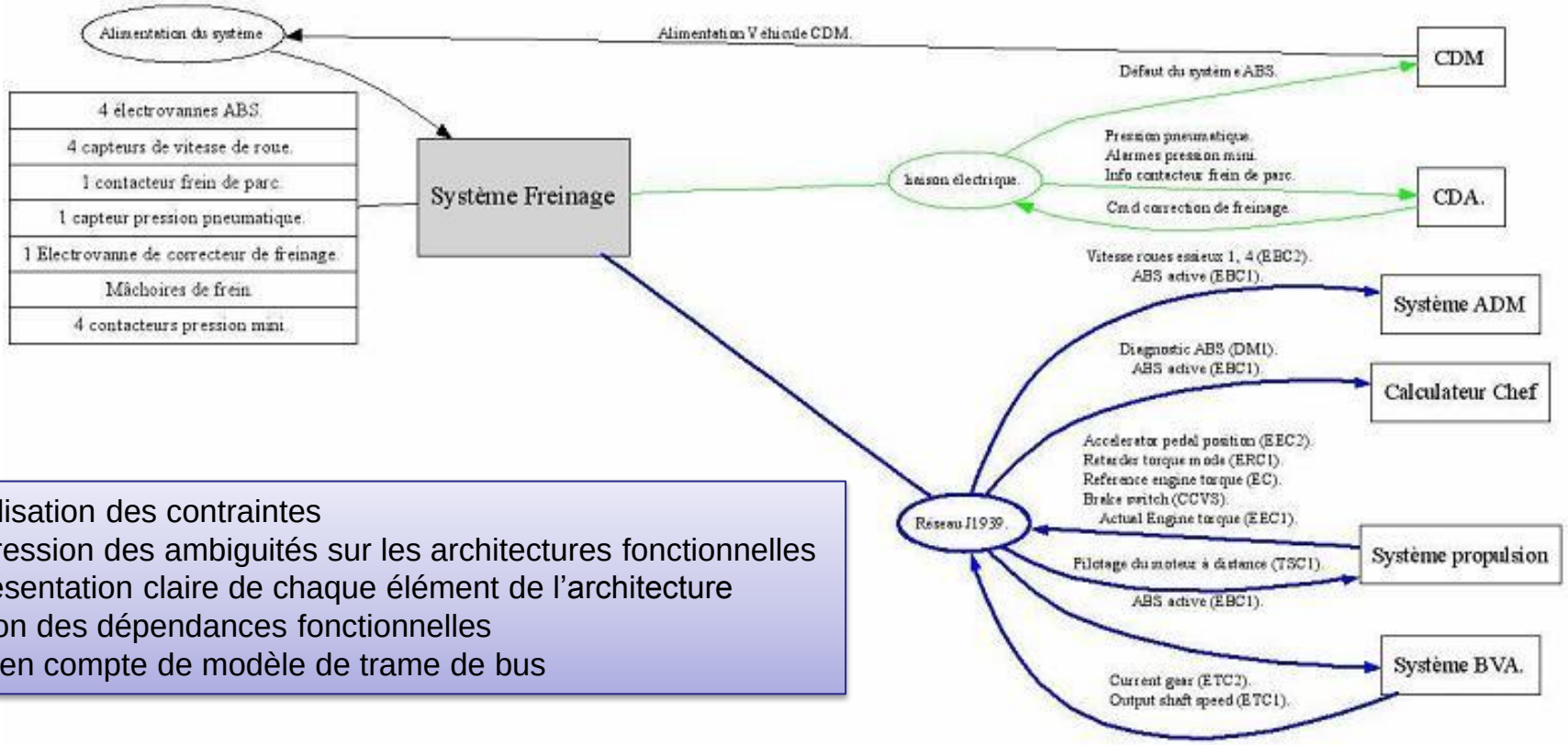
Formalisation Fonctionnelle

Pour le service Après-Vente de Peugeot

- Fonctionnalités de toute la voiture modélisées
  - Du Moteur à l'essuie glace
- Une cinquantaine de fonctionnalités majeures
- 20 personnes sur le projet sur 3 ans

Formalisation Fonctionnelle

▷ Analyse Fonctionnelle :



Modélisation des contraintes
Suppression des ambiguités sur les architectures fonctionnelles
Réprésentation claire de chaque élément de l'architecture
Gestion des dépendances fonctionnelles
Prise en compte de modèle de trame de bus

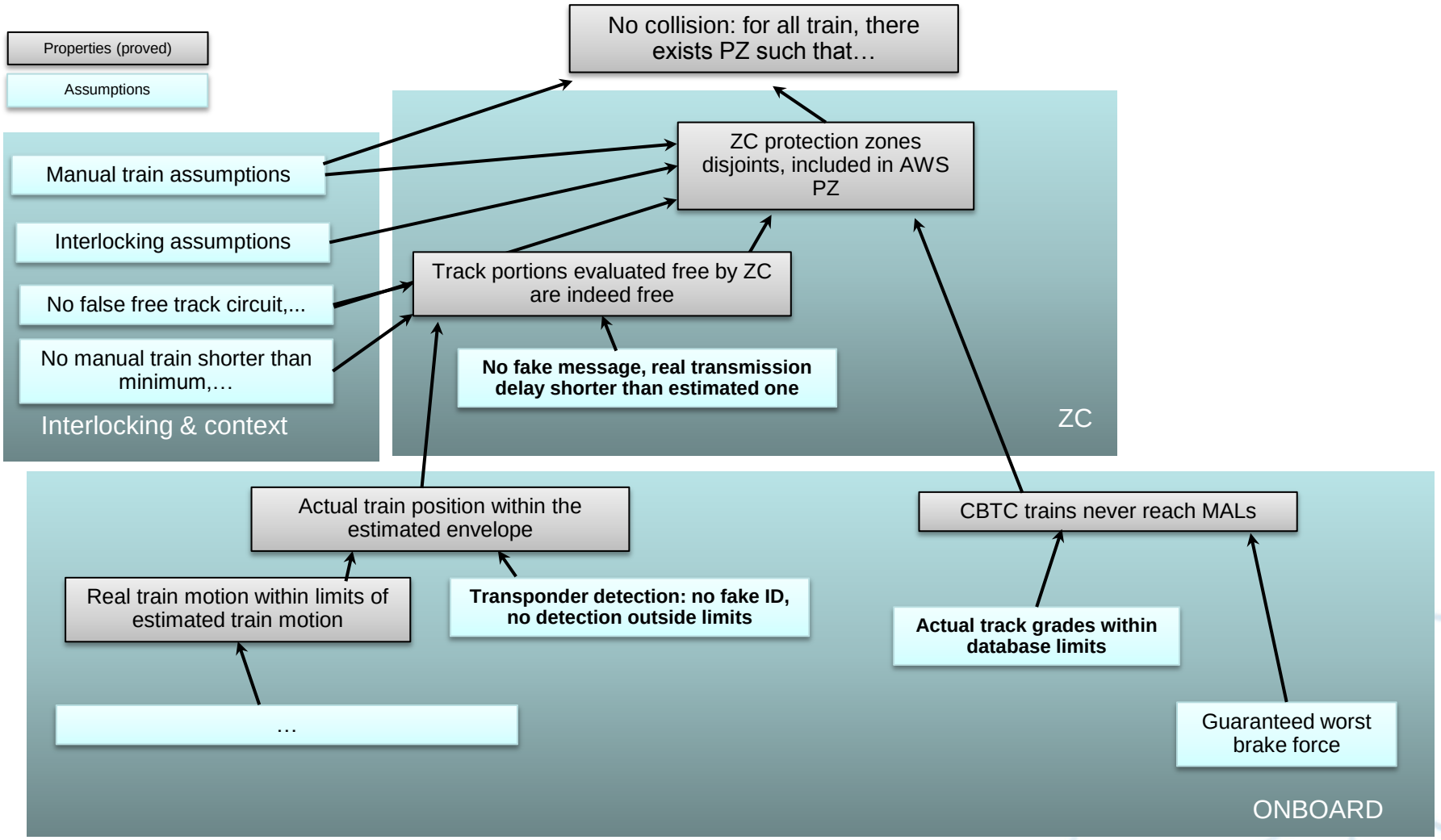
Flushing line (métro de New York)

- CBTC développé par Thalès pour le métro de NY, interopérable

  Il est possible de démontrer logiquement les propriétés nécessaires et suffisantes à la sécurité d'un grand système…

- Formalisations, raisonnements et preuves des propriétés de sûreté
  - Démonstration avec B système (+Atelier B)
    - Bénéfices / récupérabilité / utilisabilité indépendants de la méthode
- Livrables sous forme d'un document en langage naturel
  - intégrables aux Spécifications de Besoin
  - Produire des éléments de vérification adaptés à la conception de l'industriel

Propriétés traitées dans le projet Flushing Line



Développement logiciel

Développement logiciel

- Modéliser de façon abstraite le comportement d'un programme
- Raffinements successifs jusqu'à un modèle concret transcodable
- Preuve de consistance et de raffinement
- Génération de code cible.

Actuellement : Pourquoi des MF en logiciel

Développement d'un logiciel de sûreté et la preuve **pas plus cher** qu'un développement classique et test

- Confiance accrue.
  - La preuve est valable pour toutes les valeurs possibles des variables
  - Gain en terme de qualification du logiciel
  - Gain en terme de sûreté de fonctionnement (propriété garantie)

DL : erreur de programmation : Aig

Détection d'erreur de programmation

Bonne définition

- **Division par 0**
- **Fonction appliquée sur son domaine**
- **….**

Gestion Mémoire

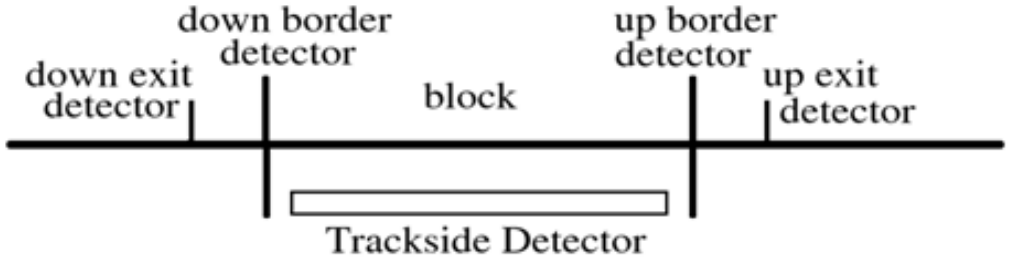
- **Boucle infinie**
- **Débordement de tableau**
- ….

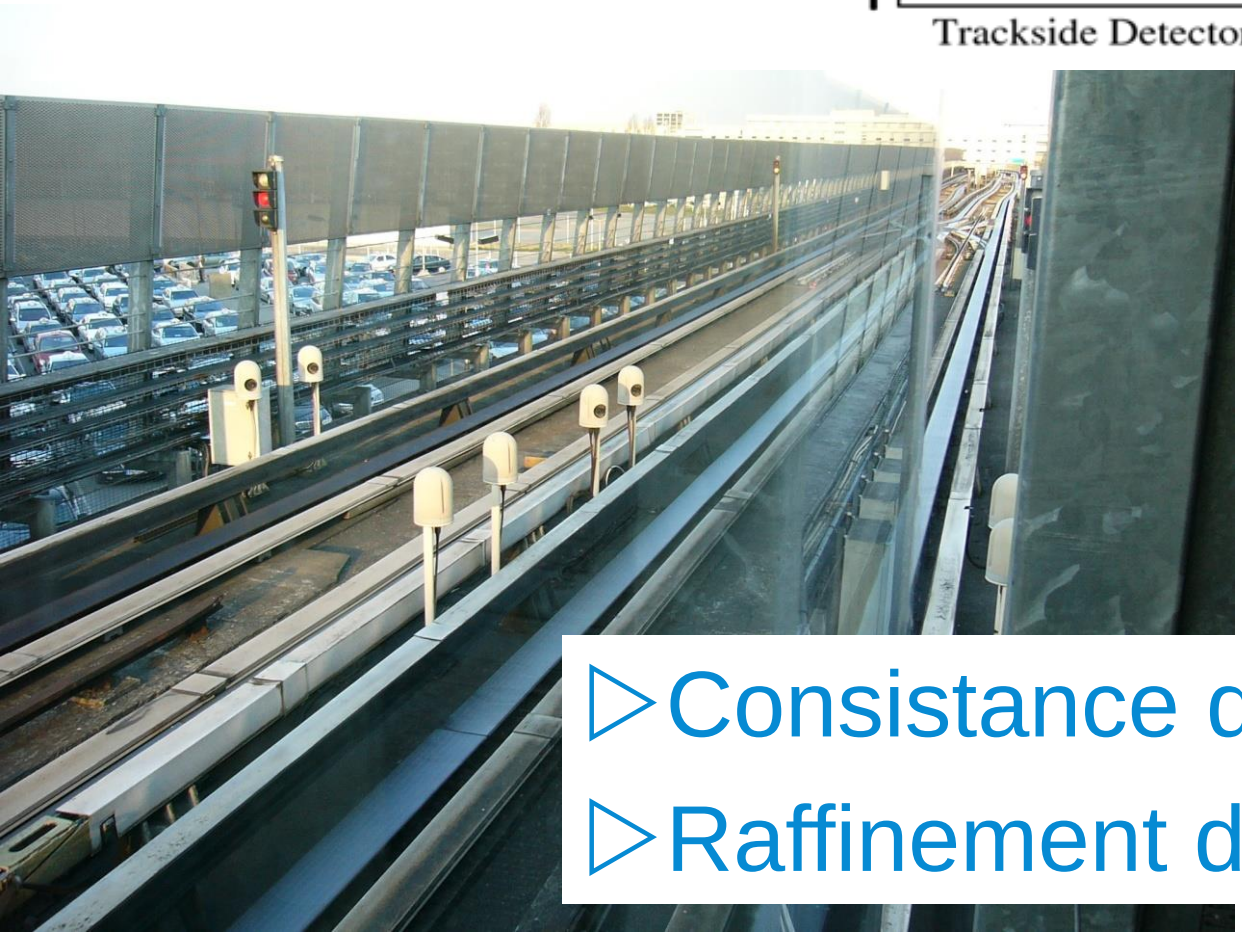
Erreur de coquille

- **Copier/coller**
- **….**

Exemple Aig

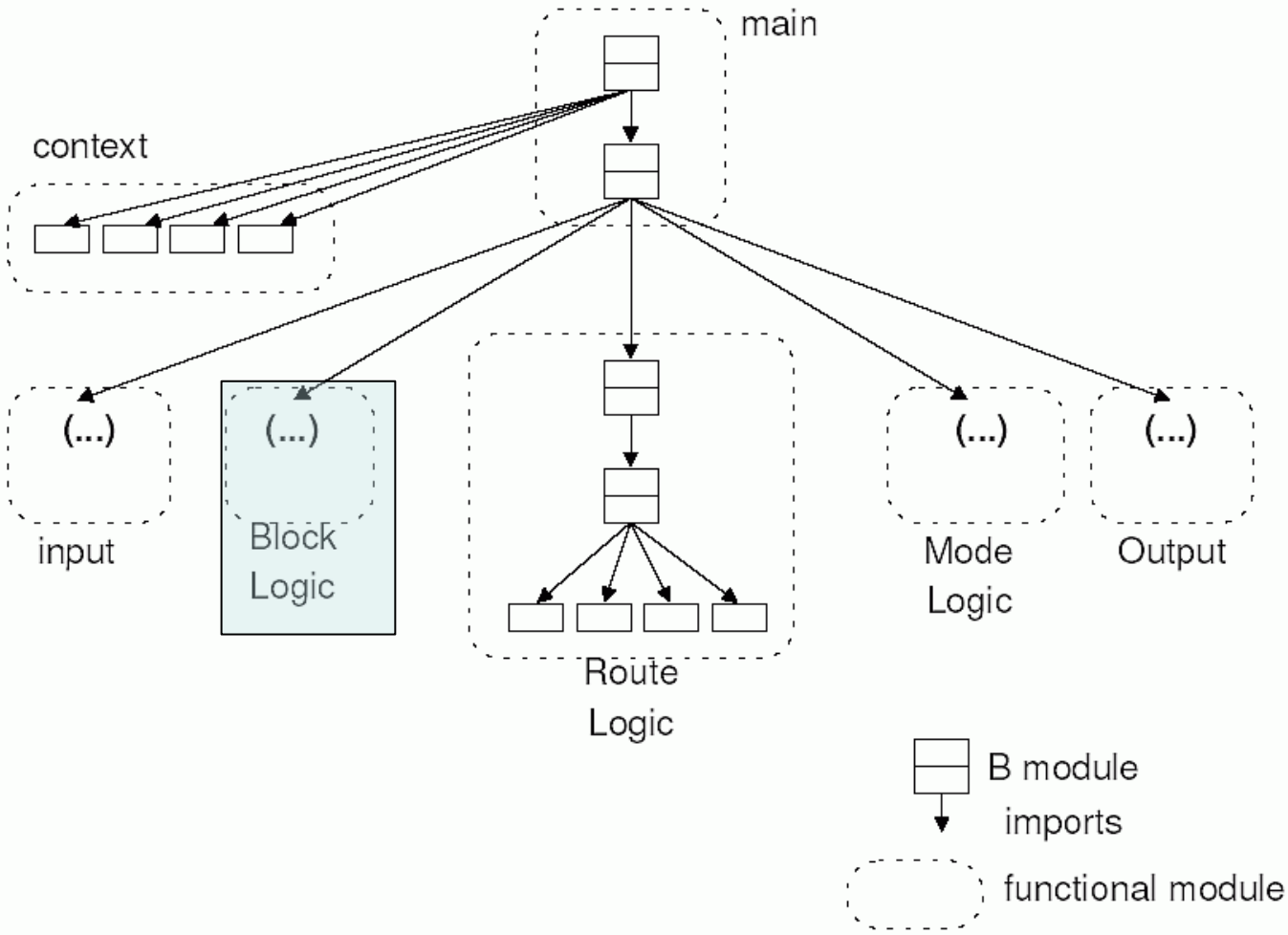
DL: Propriété : Block



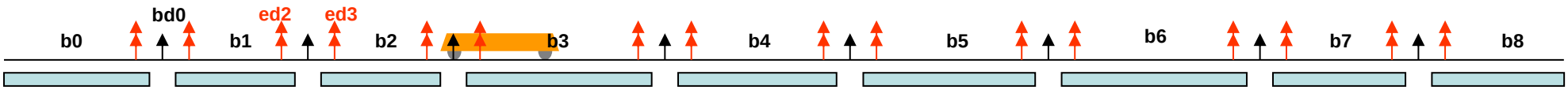


- Consistance du modèle
- Raffinement du modèle

DL: Propriété : Block



Développement logiciel : Block



configuration

t_block_i, t_border_i, t_exit_i
t_block, t_border, t_exit

cfg_b2b_up : prochain block up
cfg_b2b_down : prochain block down
cfg_b2bd_up : border u
block
cfg_b2bd_down : borde
associé au block
cfg_b2ed_up : exit up a
block
cfg_b2ed_down : exit d
block

read_all_<nom constante>

inputs

otd : blocks dont le track detector est occupé
obd : borders occupés
oed : exits occupés

booléen ← is_exit_newly_released(ed)

block_occupancy

ob : blocks occupés
mb : blocks masqués (pour TDL alarm)
tdla : blocks en alarme

unmask_blocks :
mb := mb - <blocks libres ou ceux avec tous les
TD libre>

upés, non masqué, dont le

mask_blocks :
mb := mb ∪ <blocks avec un BD occupé, sauf si le block est en alarme>

release_blocks :
ob := ob - <block (libérés par PCC ou sans alarme) qui de plus ont un ED qui a vu un front descendant>

occupy_blocks :
ob := ob ∪ <blocks avec le TD ou l'un des BD occupé>

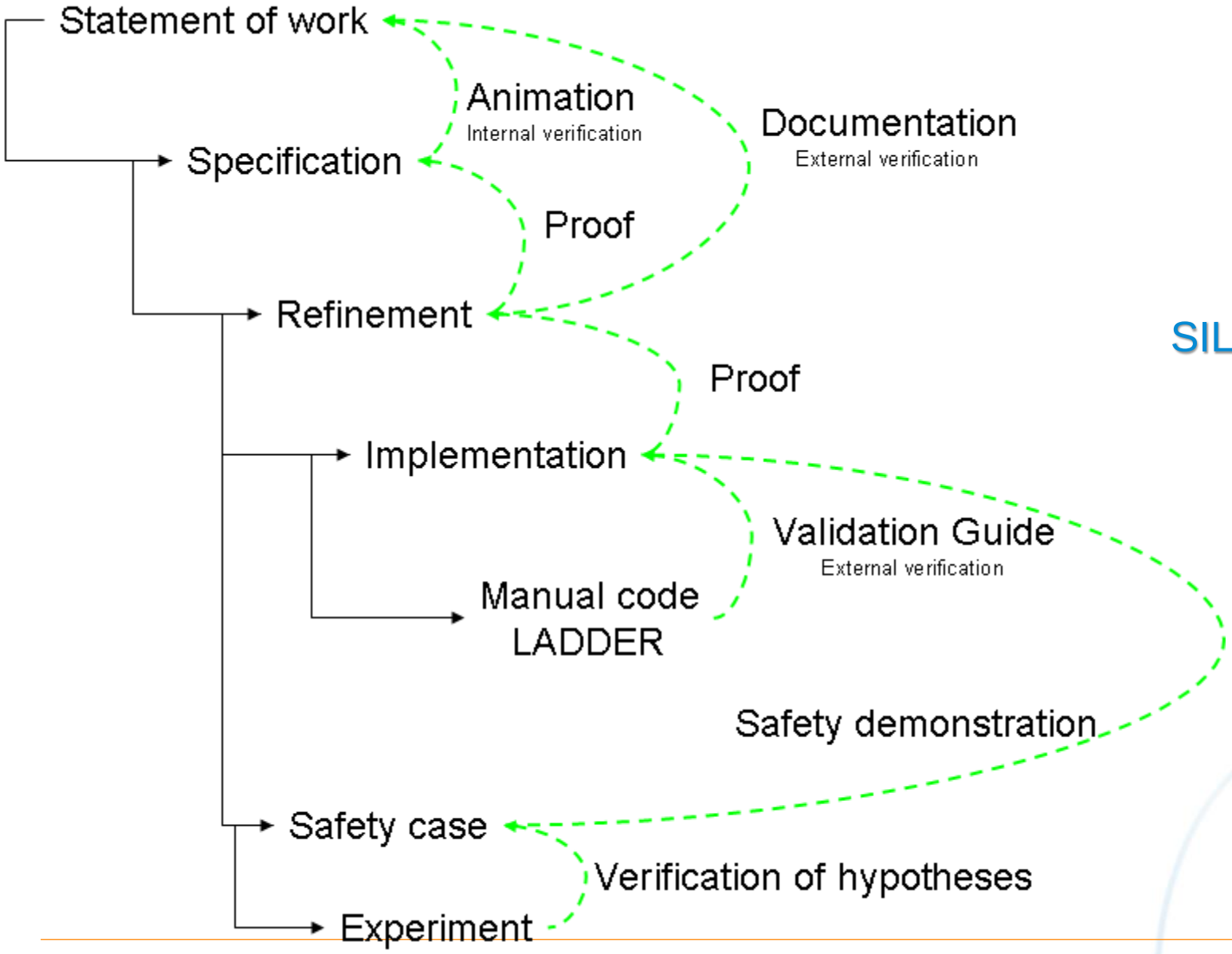
A block having one of its border detector occupied or having its trackside detector occupied has to be occupied.

Some (internal) experimentations

- Platform screen doors
- Safety critical systems (SIL3/SIL4)
- Opening and closing doors

- Event-B system level specification
- PLC code generated
- Specific to a PLC (Siemens S7)

Development cycle



Qualified for
SIL3/SIL4 systems

Validation de données

Validation de données

Etat de l'art

- Plusieurs projets de R&D ont permis de développer les outils
- Utilisation courante dans le monde industriel : Alstom, RATP, Siemens, GE

Entrées = données + règles à vérifier

Sortie = rapport de vérification en nommant les données qui ne respectent pas les règles.

Validation de données

- Formalisation des règles de transformation (génération de données)
- Formalisation de la consistance d'un ensemble de données
- Formalisation des règles de vérification (validation de données)

Validation de données

- Valider les étapes de la génération de données
- Vérifier les règles de sûreté sur les données générées
- Réduire les coûts de la validation en automatisant ces vérifications
- Respecter un processus safety
- Créer un outil adapté au processus client et aux entrées

Validation de données

Règles simples, vérification :

- de la consistance
- des constantes
- d'un champ particulier
- vérification d'unicité
- d'égalité
- etc…

Règles complexes:

- parcours de route
- parcours de distances
- opérations arithmétiques
- boucles imbriquées
- etc…

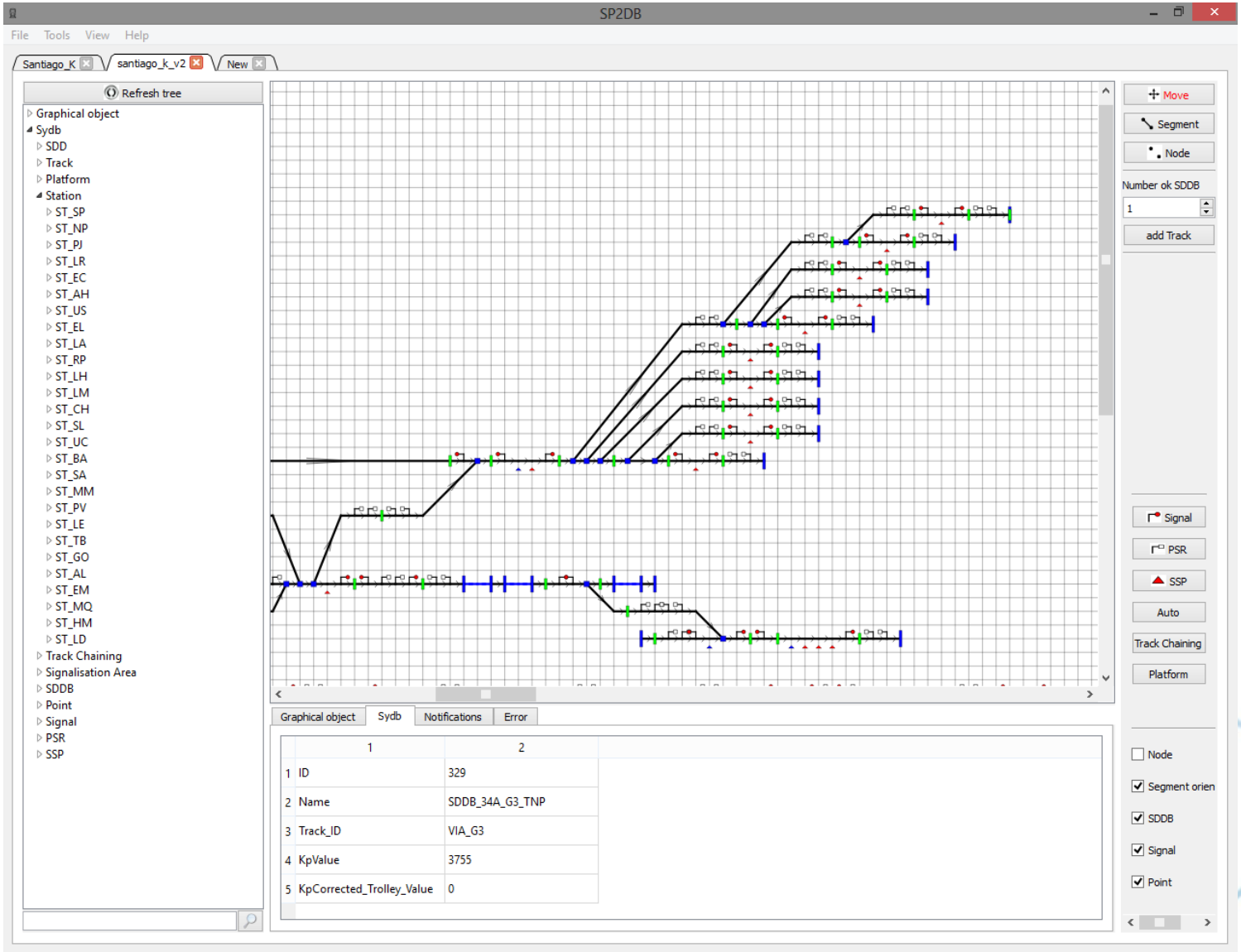
Différents outils suivant le besoin client

Les besoins clients sont différents

- Outil adapté en terme de :
  - HIM
  - Processus (double chaîne si besoin)
  - Gestion des entrées/sorties
  - Gestion des formats des données
  - Exigences et contraintes particulières

Visualisation

- Visualisation globale de la base de données
- Visualiser un objet et son environnement
- Visualiser les caractéristiques d'un objet
- Visualiser les constructions intermédiaires (overlap, sig area)



Visualisation

- Aider les ingénieurs de la génération de données :
  - Visualiser les erreurs introduites
  - Visualiser les données qui ne respectent pas les règles définies
- Favorise l'échange Systémier/formalisateur :
  - Support de communication (schéma)
  - Analyse de faux KO
- Aider les testeurs
  - Construction des bases de tests
  - Vérification du test

Ferroviaires : Chiffres types

- 800 règles modélisées et validées
- 400 constructions intermédiaires
- 50 templates
- 2000 tests créés et relancés chaque jour

- 4 fichiers XML (d'environ 50k lignes de données) validés

- Campagne de validation totale (1 système, 3 sous-systèmes) = 24 h

Conclusion

Conclusion

- 20 ans d'utilisation des MF
  - De gros projets réussis (Meteor L14)
  - Changement de la perception
  - Utilisation en augmentation
- L'utilisation des MF:
  - Applicable à différents niveaux
  - Sujets traités de plus en plus divers

Conclusion

Méthode B

Application sur des gros systèmes

- System : NY cities, Peugeot ….
- Logiciel : Meteor (L14) mais aussi plus de 50 lignes
- Validation DATA : Alstom, Siemens, GE

Pas plus cher qu'un autre

- System : coût 75
- Logiciel : équipe et temps équivalent à un développement sans MF
- Validation DATA : coût initial 2 projets

Amélioration des outils toujours en cours

- Amélioration du taux de preuve automatique (10%)
- BART : raffinement et preuve
- Outils spécifiques pour la Validation de DATA

Conclusion

Merci de votre attention