Formal Methods at Work

M. Carro

IMDEA Software Institute
and
Technical University of Madrid

May 11, 2015

Part I

Where Do We Stand?

Software is omnipresent in everyday life

Today's car: typically 100 processing units, 100 M. lines of code, 600 *Quijotes*, 7.000 programmer years.

Software is omnipresent in everyday life

Plane: computers manage controls, calculate routes, ...

Software is omnipresent in everyday life

Large interconnected systems: independent, isolated, automatic decision making, which must be globally correct.

Software is omnipresent in everyday life

- Cell phones (from O.S. to compression algorithms to user interfaces).
- HDTV (routing, encoding and decoding).
- Buy and sell on the Internet (web interfaces, databases, encryption).
- Stock market (algorithmic trading, high frequency trading).

- √ Managed by extremely complex software.
- √ All of them *critical* to a certain degree.
- √ Some **extremely** critical

- √ Managed by extremely complex software.
- √ All of them *critical* to a certain degree.
- √ Some **extremely** critical

Challenge:

How to develop complex software, with resources that are always limited, assuring that it will work correctly?

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

iOS, Mac app crashes linked to botched FairPlay DRM.

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

iOS, Mac app crashes linked to botched FairPlay DRM.

Still infected, 300,000 PCs to lose Internet access.

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

iOS, Mac app crashes linked to botched FairPlay DRM.

Still infected, 300,000 PCs to lose Internet access.

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

iOS, Mac app crashes linked to botched FairPlay DRM.

Still infected, 300,000 PCs to lose Internet access.

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

German security experts find major flaw in credit card terminals.

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

iOS, Mac app crashes linked to botched FairPlay DRM.

Still infected, 300,000 PCs to lose Internet access.

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

German security experts find major flaw in credit card terminals.

Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

- July 16, 2012: Skype bug sends messages to unintended recipients.
- July 13, 2012: Apple's "in-app purchase" service for iOS bypassed by Russian hacker.
- July 12, 2012: Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- July 5, 2012: iOS, Mac app crashes linked to botched FairPlay DRM.
- July 7, 2012: Still infected, 300,000 PCs to lose Internet access.
- July 12, 2012: Hackers expose 453,000 credentials allegedly taken from Yahoo service.
- July 13, 2012: German security experts find major flaw in credit card terminals.
- July 13, 2012: Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

- July 16, 2012: Skype bug sends messages to unintended recipients.
- July 13, 2012: Apple's "in-app purchase" service for iOS bypassed by Russian hacker.
- July 12, 2012: Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- July 5, 2012: iOS, Mac app crashes linked to botched FairPlay DRM.
- July 7, 2012: Still infected, 300,000 PCs to lose Internet access.
- July 12, 2012: Hackers expose 453,000 credentials allegedly taken from Yahoo service.
- July 13, 2012: German security experts find major flaw in credit card terminals.
- July 13, 2012: Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

Just two weeks!!!

The Ariane 5 Incident

Example: effect of a *single* integer overflow.

The Ariane 5 Incident

Example: effect of a *single* integer overflow.

From the outside...

- June 4, 1996: After launch, the Ariane 5 rocked exploded.
- This was its maiden voyage.
- It flew for about 37 sec. only in Kourou's sky.
- No injury in the disaster.

Mechanical details

- Normal behavior of the launcher for 36 sec. after lift-off.
- Failure of both Inertial Reference Systems almost simultaneously.
- Strong pivoting of the nozzles of the boosters and Vulcain engine.
- Self-destruction at an altitude of 4000 m. (1000 m. from the pad).

Mechanical details

- Normal behavior of the launcher for 36 sec. after lift-off.
- Failure of both Inertial Reference Systems almost simultaneously.
- Strong pivoting of the nozzles of the boosters and Vulcain engine.
- Self-destruction at an altitude of 4000 m. (1000 m. from the pad).

Forensic analysis

- Both inertial computers failed because of overflow on one variable.
- This caused a software exception and stops these computers.
- These computers sent post-mortem info through the bus.
- Normally the main computer receives velocity info through the bus.
- The main computer was confused and pivoted the nozzles.

But, why?

- The faulty program was working correctly on Ariane 4.
- The faulty program was not tested for A5 (since it worked for A4).
- But the velocity of Ariane 5 is far greater than that of Ariane 4.
- That caused the overflow in one variable.
- The faulty program happened to be **useless** for Ariane 5.

But, why?

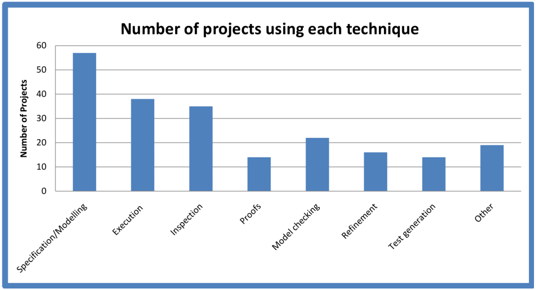
- The faulty program was working correctly on Ariane 4.
- The faulty program was not tested for A5 (since it worked for A4).
- But the velocity of Ariane 5 is far greater than that of Ariane 4.
- That caused the overflow in one variable.
- The faulty program happened to be **useless** for Ariane 5.

Message

- Does the product conform to specifications?
- Maybe it’s important to carry experience in formal methods to industry!
- Is it done?

Formal Methods in Industry

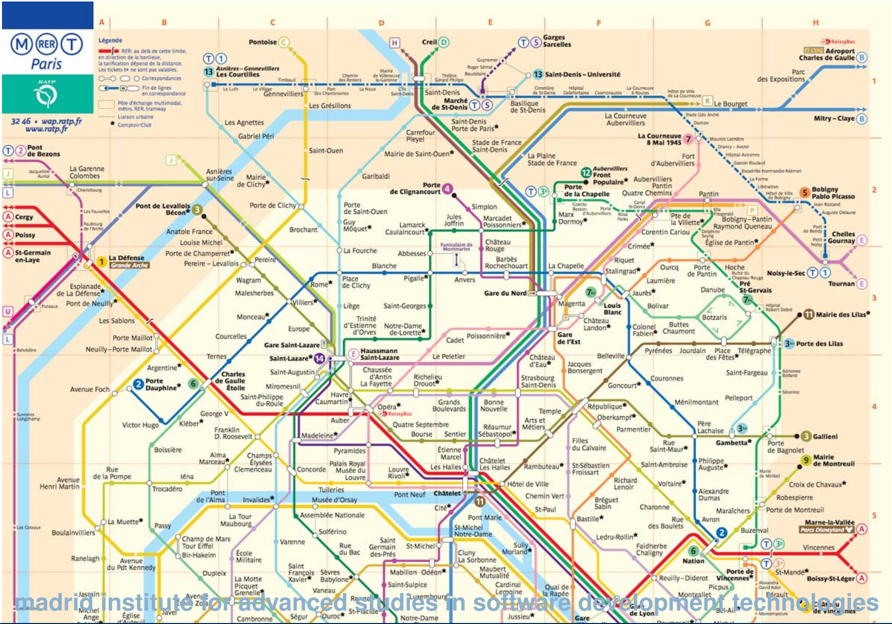
- Actually, yes — at least in some domains, for some types of applications.
- Using many approaches, actually [WLBF09].



- We will see some examples

Part II

The General Does Not Need a Driver [BBFM99]


M RER T Paris
RATP
32 46 • wap.ratp.fr
www.ratp.fr
Légende
Pontoise
Creil
Garges Sarcelles
Aéroport Charles de Gaulle
Parc des Expositions
Asnières – Gennevilliers Les Courtiles
Saint-Denis – Université
Gennevilliers
Les Grésillons
Saint-Denis
Marché de St-Denis
Basilique de St-Denis
Le Bourget
Mitry – Claye
La Courneuve Aubervilliers
Les Agnettes
Gabriel Péri
Carrefour Pleyel
Saint-Denis Porte de Paris
Stade de France Saint-Denis
La Courneuve 8 Mai 1945
Fort d'Aubervilliers
Saint-Ouen
Mairie de Saint-Ouen
La Plaine Stade de France
Aubervilliers Front Populaire
Aubervilliers Pantin Quatre Chemins
Pont de Bezons
La Garenne Colombes
Asnières sur-Seine
Mairie de Clichy
Garibaldi
Porte de Clignancourt
Porte de la Chapelle
Pantin
Bobigny Pablo Picasso
Cergy
Poissy
St-Germain en-Laye
Pont de Levallois Bécon
Porte de Clichy
Porte de Saint-Ouen
Simplon
Marcadet Poissonniers
Marx Dormoy
Porte de la Villette
Bobigny – Pantin Raymond Queneau
Chelles Gournay
Anatole France
Louise Michel
Brochant
Guy Môquet
Jules Joffrin
Lamarck Caulaincourt
Corentin Cariou
Crimée
Église de Pantin
La Défense Grande Arche
Porte de Champerret
Pereire – Levallois
Pereire
La Fourche
Abbesses
Château Rouge
Barbès Rochechouart
La Chapelle
Riquet
Stalingrad
Ourcq
Porte de Pantin
Hoche
Noisy-le-Sec
Tournan
Esplanade de La Défense
Wagram
Rome
Place de Clichy
Blanche
Pigalle
Anvers
Gare du Nord
Magenta
Jaurès
Laumière
Danube
Pré St-Gervais
Pont de Neuilly
Malesherbes
Villiers
Liège
Saint-Georges
Louis Blanc
Bolivar
Botzaris
Mairie des Lilas
Les Sablons
Monceau
Europe
Trinité d'Estienne d'Orves
Notre-Dame de-Lorette
Château Landon
Colonel Fabien
Buttes Chaumont
Porte Maillot
Courcelles
Gare Saint-Lazare
Cadet
Poissonnière
Gare de l'Est
Porte des Lilas
Neuilly – Porte Maillot
Ternes
Saint-Lazare
Haussmann Saint-Lazare
Le Peletier
Château d'Eau
Belleville
Pyrénées
Jourdain
Place des Fêtes
Télégraphe
Argentine
Charles de Gaulle Étoile
Saint-Augustin
Chaussée d'Antin La Fayette
Richelieu Drouot
Strasbourg Saint-Denis
Jacques Bonsergent
Couronnes
Saint-Fargeau
Avenue Foch
Porte Dauphine
Miromesnil
Saint-Philippe du-Roule
Havre Caumartin
Opéra
Grands Boulevards
Bonne Nouvelle
Goncourt
Pelleport
Ménilmontant
George V
Auber
Quatre Septembre
Temple
République
Victor Hugo
Kléber
Franklin D. Roosevelt
Madeleine
Bourse
Sentier
Réaumur Sébastopol
Arts et Métiers
Oberkampf
Parmentier
Père Lachaise
Gambetta
Gallieni
Porte de Bagnolet
Boissière
Pyramides
Étienne Marcel
Filles du Calvaire
Rue Saint-Maur
Alma Marceau
Champs Élysées Clemenceau
Palais Royal Musée du Louvre
Louvre Rivoli
Les Halles
Châtelet Les Halles
Rambuteau
St-Sébastien Froissart
Saint-Ambroise
Philippe Auguste
Mairie de Montreuil
Rue de la Pompe
Avenue Henri Martin
Iéna
Concorde
Hôtel de Ville
Richard Lenoir
Voltaire
Croix de Chavaux
Trocadéro
Tuileries
Pont Neuf
Châtelet
Chemin Vert
Alexandre Dumas
Robespierre
Pont de l'Alma
Invalides
Musée d'Orsay
Cité
St-Paul
Bréguet Sabin
Charonne
Maraîchers
Porte de Montreuil
La Muette
Passy
La Tour Maubourg
Assemblée Nationale
Pont Marie
Bastille
Rue des Boulets
Avron
Marne-la-Vallée
Boulainvilliers
Champ de Mars Tour Eiffel
Bir-Hakeim
Solférino
St-Michel Notre-Dame
Ledru-Rollin
Buzenval
Varenne
Saint Germain des-Prés
St-Michel
Sully Morland
Faidherbe Chaligny
Vincennes
Ranelagh
Avenue du Pdt Kennedy
École Militaire
Rue du Bac
Cluny La Sorbonne
Nation
Dupleix
Saint François Xavier
Sèvres Babylone
Mabillon
Odéon
Maubert Mutualité
Reuilly – Diderot
Porte de Vincennes
St-Mandé
Boissy-St-Léger
La Motte Picquet Grenelle
Vaneau
Saint-Sulpice
Cardinal Lemoine
Quai de la Rapée
Picpus
Montgallet
Jasmin
Michel Ange
Javel
Église d'Auteuil
Cambronne
Ségur
Duroc
St-Placide
Notre-Dame
Jussieu
Gare de Lyon
Bel-Air
Michel
Daumesnil

Driving a Subway Train Is Easy. . . Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, . . .
- Should be easy to automate!
- But...
  - Wait for people to get on board.

[1] Same with trains, only worse.

Driving a Subway Train Is Easy. . . Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, . . .
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.

[1]Same with trains, only worse.

Driving a Subway Train Is Easy. . . Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, . . .
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don't get too close to the next train.

[1] Same with trains, only worse.

Driving a Subway Train Is Easy. . . Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, . . .
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don't get too close to the next train.
  - How can you to try to see if it works?[1]

[1] Same with trains, only worse.

Driving a Subway Train Is Easy. . . Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, . . .
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don't get too close to the next train.
  - How can you to try to see if it works?[1]
  - And, if it doesn't, how long does it take to reproduce the bug in your computer?

[1] Same with trains, only worse.

Driving a Subway Train Is Easy. . . Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, . . .
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don't get too close to the next train.
  - How can you to try to see if it works?[1]
  - And, if it doesn't, how long does it take to reproduce the bug in your computer?
  - Or, plainly, how can you reproduce it?

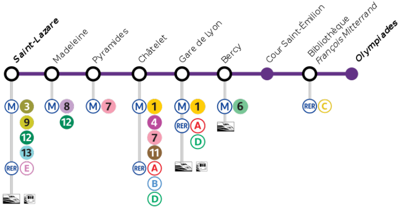
[1] Same with trains, only worse.

Towards an Automatically Managed Subway Line

- Add extra sensors.
- Add extra mechanisms (doors apart apart from coaches').
- Model the enviroment very precisely.
- Correctness by Construction.

Paris Metro Line 14

- October 15th, 1998, Tolbiac to Madeleine. (Now extended)
- 40000 passenger / hour, 85 sec. between trains in peak hour.
- 2009: 60 million passengers / year.
- Decision based on previous experience:
  - Completely automated line.
  - Completely developed using formal methods for control systems.
  - Having manually and automatially driven trains.



Subsystems

- Automatic control and signaling.
- Platform doors.
- Audio and Video.
- Operating control center.

Subsystems

- **Automatic control and signaling**.
  - In the train.
  - In the operating control center.
  - Along the tracks.
- Platform doors.
- Audio and Video.
- Operating control center.

Subsystems

- **Automatic control and signaling**.
  - In the train.
  - In the operating control center.
  - Along the tracks.
  - Running alone on specific, non-interruptible, microprocessor boards.
- Platform doors.
- Audio and Video.
- Operating control center.

Subsystems

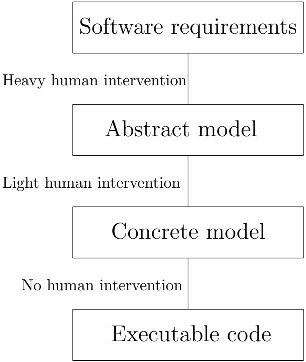
- **Automatic control and signaling**.
  - In the train.
  - In the operating control center.
  - Along the tracks.
  - Running alone on specific, non-interruptible, microprocessor boards.
- Platform doors.
- Audio and Video.
- Operating control center.
- Developed using the *B Method* [Abr96], now evolved into Event B [Abr10].

Correctness by construction

Make sure that every step is correct

A Taste of B

Model	the problem.
Determine	the properties to hold.
Refine	the model (several times, adding more details).
Prove	that the refinement is correct.



Correctness by construction

Make sure that every step is correct

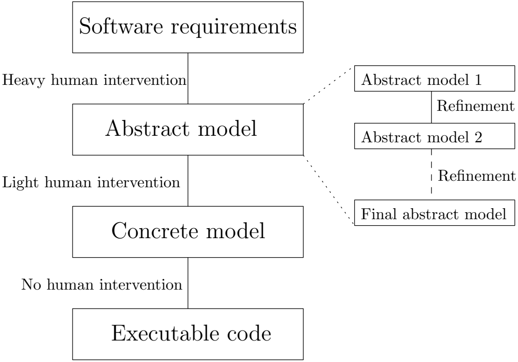
A Taste of B

Model the problem.

Determine the properties to hold.

Refine the model (several times, adding more details).

Prove that the refinement is correct.



Correctness by construction

Make sure that every step is correct

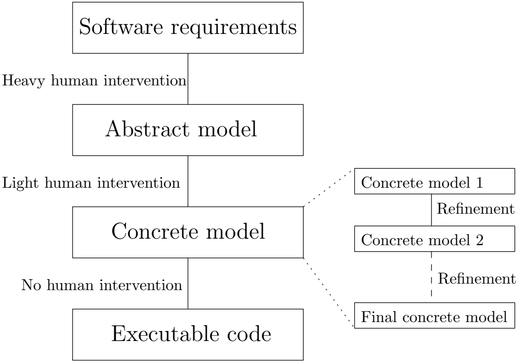
A Taste of B

Model the problem.

Determine the properties to hold.

Refine the model (several times, adding more details).

Prove that the refinement is correct.



Correctness by construction

Make sure that every step is correct

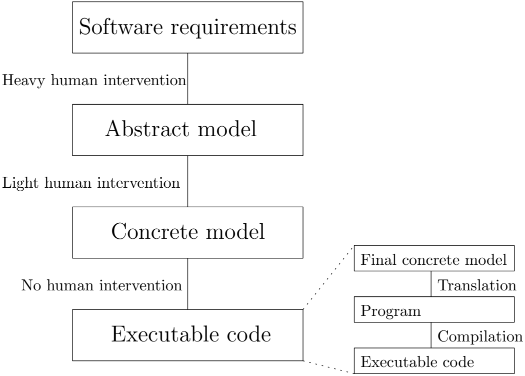
A Taste of B

Model the problem.

Determine the properties to hold.

Refine the model (several times, adding more details).

Prove that the refinement is correct.



INVARIANTS

`invBDay` : $birthday \in PERSON \mapsto DATE$

EVENTS

Initialisation

begin

`initBDay` : $birthday := \varnothing$

end

Event *addBDay* $\widehat{=}$

any

p, d

where

`inPerson` : $p \in PERSON$

`inDate` : $d \in DATE$

then

`newBDay` : $birthday(p) := d$

end

END

INVARIANTS

`invBDay` : $birthday \in PERSON \nrightarrow DATE$

EVENTS

Initialisation

begin

`initBDay` : $birthday := \varnothing$

end

Event *addBDay* $\widehat{=}$

any

p, d

where

`inPerson` : $p \in PERSON$

`inDate` : $d \in DATE$

`checkBday` : $p \notin dom(birthday)$

then

`newBDay` : $birthday(p) := d$

end

END

Proofs: Prove **mathematically** refinements are right.

Proofs: Prove **mathematically** refinements are right.

$$\dfrac{\dfrac{\dfrac{}{n > 0 \vdash 0 = 0}\ \text{EQL} \qquad \dfrac{}{n > 0 \vdash n > 0}\ \text{HYP}}{n > 0 \vdash (n > 0 \wedge 0 = 0)}\ \text{AND\_R}}{n > 0 \vdash (n < d \wedge 0 = 0) \vee (n > 0 \wedge 0 = 0)}\ \text{OR\_R}$$

$$\dfrac{\dfrac{\dfrac{}{n < d \vdash 0 = 0}\ \text{EQL} \qquad \dfrac{}{n < d \vdash n < d}\ \text{HYP}}{n < d \vdash (n < d \wedge 0 = 0)}\ \text{AND\_R}}{n < d \vdash (n < d \wedge 0 = 0) \vee (n > 0 \wedge 0 = 0)}\ \text{OR\_R}$$

$$\dfrac{n > 0 \vdash (n < d \wedge 0 = 0) \vee (n > 0 \wedge 0 = 0) \qquad n < d \vdash (n < d \wedge 0 = 0) \vee (n > 0 \wedge 0 = 0)}{n > 0 \vee n < d \vdash (n < d \wedge 0 = 0) \vee (n > 0 \wedge 0 = 0)}\ \text{OR\_L}$$

$$\dfrac{n > 0 \vee n < d \vdash (n < d \wedge 0 = 0) \vee (n > 0 \wedge 0 = 0)}{b = n, n > 0 \vee n < d \vdash (b < d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)}\ \text{EQ\_LR}$$

$$\dfrac{b = n, n > 0 \vee n < d \vdash (b < d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)}{0 + b + 0 = n, n > 0 \vee n < d \vdash (0 + b < d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)}\ \text{ARITH}$$

$$\dfrac{0 + b + 0 = n, n > 0 \vee n < d \vdash (0 + b < d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)}{a + b + 0 = n, n > 0 \vee n < d, a = 0 \vdash (a + b < d \wedge 0 = 0) \vee (b > 0 \wedge a = 0)}\ \text{EQ\_LR}$$

$$\dfrac{a + b + 0 = n, n > 0 \vee n < d, a = 0 \vdash (a + b < d \wedge 0 = 0) \vee (b > 0 \wedge a = 0)}{a \in \mathbb{N}, a + b + 0 = n, n > 0 \vee n < d, \neg(a > 0) \vdash (a + b < d \wedge 0 = 0) \vee (b > 0 \wedge a = 0)}\ \text{ARITH}$$

$$\dfrac{a \in \mathbb{N}, a + b + 0 = n, n > 0 \vee n < d, \neg(a > 0) \vdash (a + b < d \wedge 0 = 0) \vee (b > 0 \wedge a = 0)}{a \in \mathbb{N}, a + b + 0 = n, n > 0 \vee n < d \vdash (a + b < d \wedge 0 = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}\ \text{NEG}$$

$$\dfrac{a \in \mathbb{N}, a + b + 0 = n, n > 0 \vee n < d \vdash (a + b < d \wedge 0 = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}{a \in \mathbb{N}, a + b + c = n, n > 0 \vee n < d, c = 0 \vdash (a + b < d \wedge c = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}\ \text{EQ\_LR}$$

$$\dfrac{a \in \mathbb{N}, a + b + c = n, n > 0 \vee n < d, c = 0 \vdash (a + b < d \wedge c = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}{a, c \in \mathbb{N}, a + b + c = n, n > 0 \vee n < d, \neg(c > 0) \vdash (a + b < d \wedge c = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}\ \text{ARITH}$$

$$\dfrac{a, c \in \mathbb{N}, a + b + c = n, n > 0 \vee n < d, \neg(c > 0) \vdash (a + b < d \wedge c = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}{a, c \in \mathbb{N}, a + b + c = n, n > 0 \vee n < d \vdash (a + b < d \wedge c = 0) \vee c > 0 \vee a > 0 \vee (b > 0 \wedge a = 0)}\ \text{NEG}$$

$$\dfrac{a, c \in \mathbb{N}, a + b + c = n, n > 0 \vee n < d \vdash (a + b < d \wedge c = 0) \vee c > 0 \vee a > 0 \vee (b > 0 \wedge a = 0)}{a, b, c, d, n \in \mathbb{N}, 0 < d, n \leq d, a + b + c = n, a = 0 \vee c = 0, 0 < n \vee n < d \vdash (a + b < d \wedge c = 0) \vee c > 0 \vee a > 0 \vee (b > 0 \wedge a = 0)}\ \text{MON}$$

Proofs: Prove **mathematically** refinements are right.

$$\dfrac{\dfrac{\dfrac{}{n>0\vdash 0=0}\text{EQL}\quad\dfrac{}{n>0\vdash n>0}\text{HYP}}{n>0\vdash(n>0\wedge 0=0)}\text{AND\_R}}{n>0\vdash(n<d\wedge 0=0)\vee(n>0\wedge 0=0)}\text{OR\_R}$$

$$\dfrac{\dfrac{\dfrac{}{n<d\vdash 0=0}\text{EQL}\quad\dfrac{}{n<d\vdash n<d}\text{HYP}}{n<d\vdash(n<d\wedge 0=0)}\text{AND\_R}}{n<d\vdash(n<d\wedge 0=0)\vee(n>0\wedge 0=0)}\text{OR\_R}$$

$$\dfrac{\dfrac{}{n>0\vee n<d\vdash(n<d\wedge 0=0)\vee(n>0\wedge 0=0)}\text{OR\_L}}{b=n,n>0\vee n<d\vdash(b<d\wedge 0=0)\vee(b>0\wedge 0=0)}\text{EQ\_LR}$$

ARITH

Note: invariants are the most important piece of information. Although many proofs are not aimed at establishing invariants, virtuall all of them involve invariants.

TH

G

$0\vee(b>0\wedge a=0)$

ARITH

$a,c\in\mathbb{N},a+b+c=n,n>0\vee n<d,\neg(c>0)\vdash(a+b<d\wedge c=0)\vee a>0\vee(b>0\wedge a=0)$

NEG

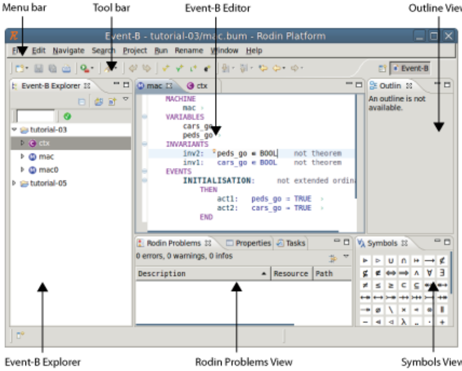
$a,c\in\mathbb{N},a+b+c=n,n>0\vee n<d\vdash(a+b<d\wedge c=0)\vee c>0\vee a>0\vee(b>0\wedge a=0)$

MON

$a,b,c,d,n\in\mathbb{N},0<d,n\leq d,a+b+c=n,a=0\vee c=0,0<n\vee n<d\vdash(a+b<d\wedge c=0)\vee c>0\vee a>0\vee(b>0\wedge a=0)$

Tools!

- Automate some (many) proofs.
- Automatically generate code.
- Many advantages when code is generated.
- Example at hand: code duplication.
    - Electronic in tunnels: interference with boards.
    - Possible corruption of data.
    - All data duplicated in different formats.
    - Code works on both copies.
    - Constant comparison for consistency.



Statistics

- **Lines of B:** 115.000 .
- **Lines of Ada:** 86.000 .
- **Lemmas of B:** 27.800 .
- **Automatically proven:** 92%.
- **Time to develop:** 4 years (aprox.)

Statistics

- **Lines of B:** 115.000 .
- **Lines of Ada:** 86.000 .
- **Lemmas of B:** 27.800 .
- **Automatically proven:** 92%.
- **Time to develop:** 4 years (aprox.)

- **Bugs in development computer:** 0
- **Bugs in target computer:** 0
- **Bugs in on-site tests:** 0
- **Bugs since deployment:** 0

Part III

Amazon Had 2.000.000.000.000 Things to Care About [NTZ+14]

Landscape

Some numbers for AWS's S3

- 2013: 2.000.000.000.000 (2 trillions) objects, 1.1 million requests per second.
- High availability.
- Scalability.
- S3 just one AWS service.
- Essential complexity high → unavoidable human errors.

Previously in...

- Standard “verification” techniques in industry.
  - Deep design reviews.
  - Code reviews (c.f. DB train code reviews).
  - Static code analysis.
  - Stress testing.
  - Fault-injection testing.
  - . . .

> *. . . human intuition is poor at estimating the true probability of supposedly “extremely rare” combinations of events in systems operating at a scale of millions of requests per second.*

The History According to the Starring Roles

- Engineer C.N. dissatisfied with bugs in implementaions of distributed algorithms.
- Looked for ways to correct them — not thinking on formal methods.
- But read paper on formal verification of Chord using Alloy.
- Tried to use it, but not rich enough — no good for formal methods!

The History According to the Starring Roles

- Engineer C.N. dissatisfied with bugs in implementaions of distributed algorithms.
- Looked for ways to correct them — not thinking on formal methods.
- But read paper on formal verification of Chord using Alloy.
- Tried to use it, but not rich enough — no good for formal methods!
- C.N. read a paper on Leslie Lamport Paxos algorithm — essential in distributed systems.
- At the end of the paper, a TLA+ [Lam02] / TLC formalization.
- TLA+ also devised by Leslie Lamport.
- Maybe TLA+ / TLC was worth something?
- Tried the same example as in Alloy — success!

The Busy Engineers

- But engineers too busy to try new things.
- Unless there is a new need — and a new need appeared.

The Busy Engineers

- But engineers too busy to try new things.
- Unless there is a new need — and a new need appeared.

Enter DynamoDB

- Scalable, high-performance, high-availability storage.
- Testing, stressing, fault injection — but really high confidence was necessary.
  - Otherwise data from companies could be lost.
- T.R. (DynamoDB's coauthor) went on to prove relevant properties.
- Informal proofs already found bugs.
- Which other subtle problems could be hidden?

TLA+ Time

- T.R. learned TLA+, formalized (some) DynamoDB algorithms.
- Run distributed version of model checker TLC.
  - Cluster of ten EC^2 instances.
  - Each 8 cores + hyperthreads.
  - 23 GB ram.
- Small part of algorithm OK.
- But in the full fault-tolerance algorithm a bug was found.
  - Very subtle — many conditions had to be met.
  - But historically possible.
  - Bug had passed all reviews.
  - Other two bugs were found later on.

The History Goes On

- New DynamoDB features first modeled and verified in TLA+ — and bugs were found ahead of time.
- Presentation to teams: *Debugging Designs*.
  - *Exahustively testable pseudo-code.*
- New fault-tolerant distributed algorithm specified and checked.
  - Two bugs found.
- Management started pushing TLA+ usage in teams.

What the Protagonists Say...

[...] help engineers to get the design right. [...] If the design is broken then the code is almost certainly broken. Coding mistakes extremely unlikely to compensate mistakes in design.

Engineers probably deceived into believing that code is ‘correct’ because appears to correctly implement the broken design. Unlikely to realize that design is incorrect while focusing on coding.

[...] gain a better understanding of the design. [...] can only increase chances that they will get code right.

[...] write better assertions [...], a good way to reduce errors in code.

Formal methods help engineers to find strong invariants, so formal methods can help to improve assertions, which help improve the quality of code.

Part IV

That Is Not What I Meant [PPS$^+$03]

A Different Problem

How misleading are (well-written) natural language specifications?

A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it's not, it relies a lot on "common knowledge" or "situational knowledge".
- Kids give us lots of examples: flawless reasoning without knowledge.

A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it's not, it relies a lot on “common knowledge” or “situational knowledge”.
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it's not, it relies a lot on "common knowledge" or "situational knowledge".
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?

A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it's not, it relies a lot on "common knowledge" or "situational knowledge".
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
- We buy another faucet.

A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it's not, it relies a lot on "common knowledge" or "situational knowledge".
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
- We buy another faucet.
- (Pointing below the sink). Look: the tap gets water from the pipe, which comes from the wall. Buying another faucet will not help. Now, what would be do?

A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it's not, it relies a lot on "common knowledge" or "situational knowledge".
- Kids give us lots of examples: flawless reasoning without knowledge.

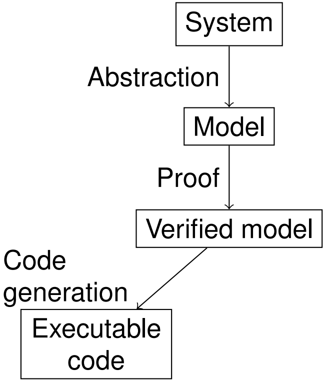
A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
- We buy another faucet.
- (Pointing below the sink). Look: the tap gets water from the pipe, which comes from the wall. Buying another faucet will not help. Now, what would be do?
- We buy another wall.

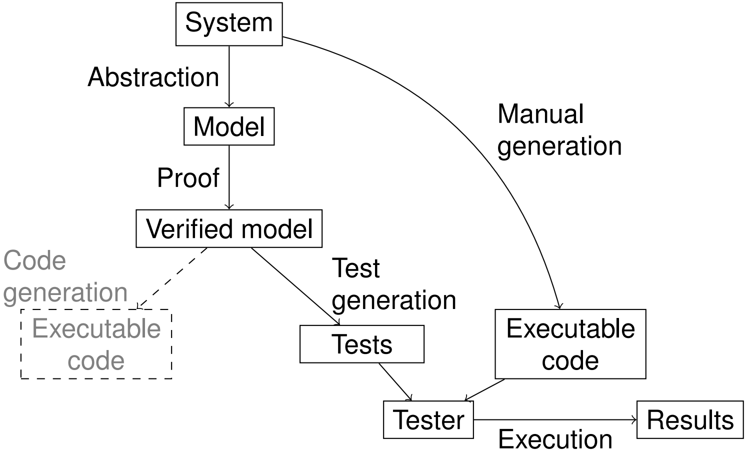
A Different Experiment

- Existing, well-proven implementation of a smartcard.
- Natural language specification.
- How well can a team of engineers capture understand these specifications?

Using Models to Generate Tests



Using Models to Generate Tests



A SmartCard is...

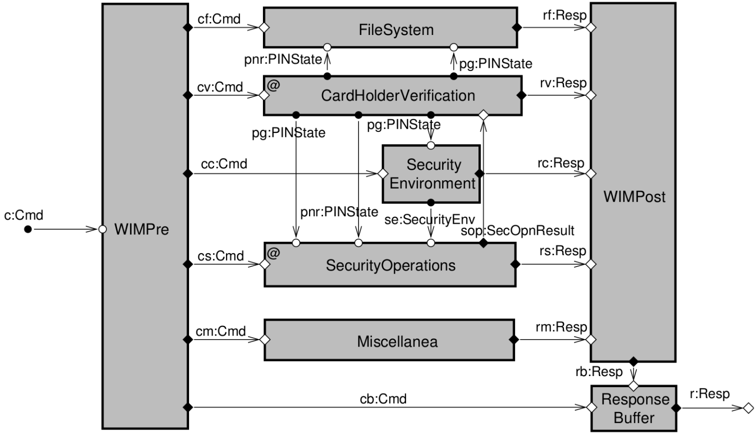
- Fully programmable one-chip computer.
- Microprocessor, RAM (currently 256-4096 Bytes), EEPROM (2-16 KBytes), and ROM (8-64 KBytes).
- Hierarchical filesystem.
- Serial interface.
- Sometimes specialized cryptographic microprocessor.

A SmartCard is...

- Fully programmable one-chip computer.
- Microprocessor, RAM (currently 256-4096 Bytes), EEPROM (2-16 KBytes), and ROM (8-64 KBytes).
- Hierarchical filesystem.
- Serial interface.
- Sometimes specialized cryptographic microprocessor.
- Command interpreter:
  - Read commands from stdin.
  - Interpret, execute, write output to stdout.
  - Execution usually depends on previous commands.

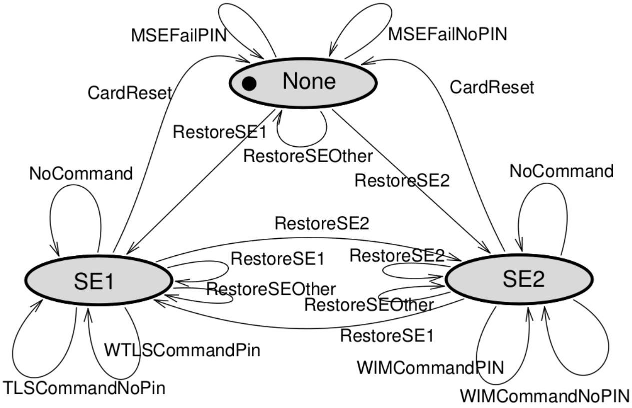
System Under Study

- WAP module in GSM standard, implemented in a SmartCard.



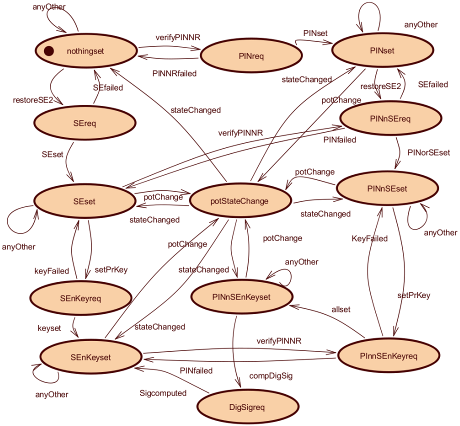
Subcomponents

- Each subcomponent has a expected behavior (i.e., what it expects to receive and return in every state).

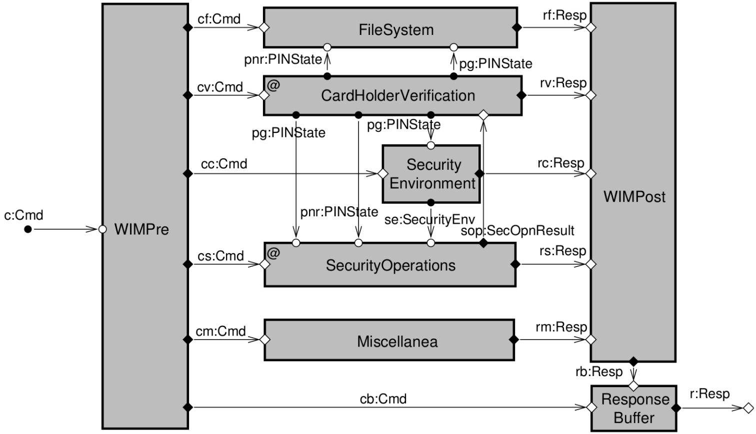


Tester: Behavior

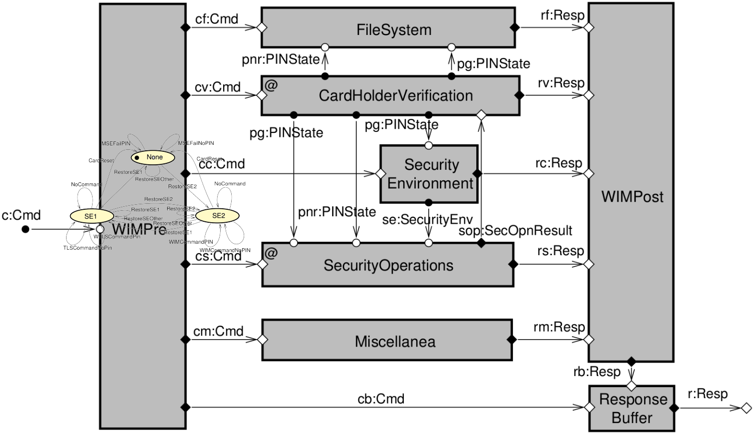
From the behavior of the component one can extract the expected behavior of the tester: what it has to do at every moment to check the component.



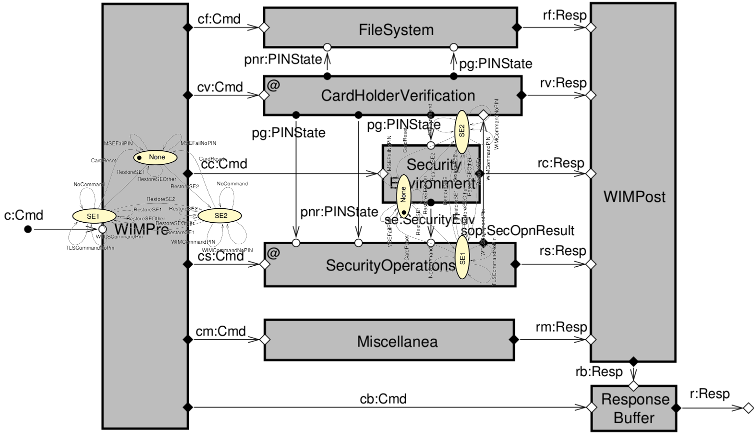
The System to Test



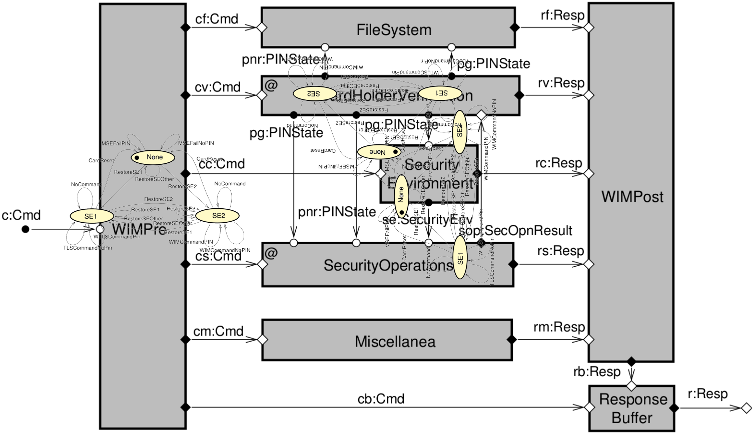
The System to Test



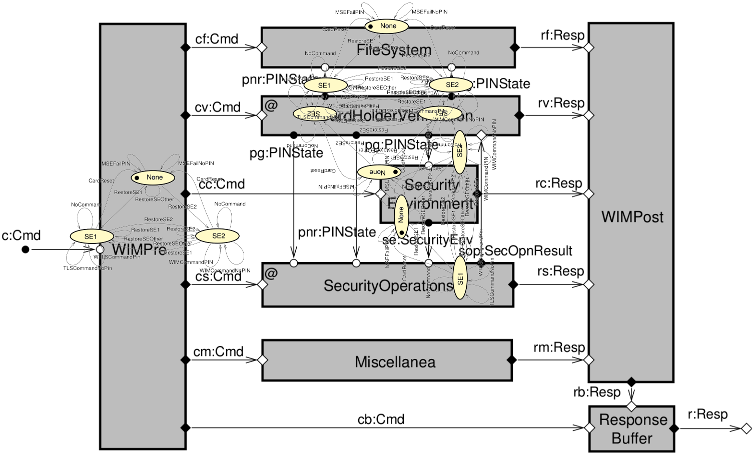
The System to Test



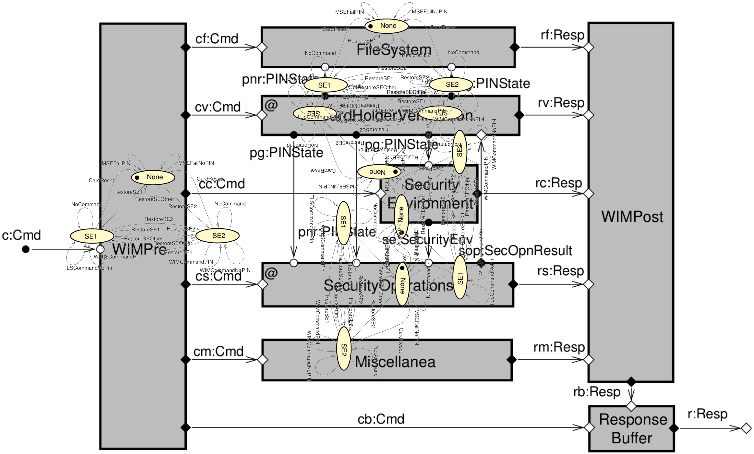
The System to Test



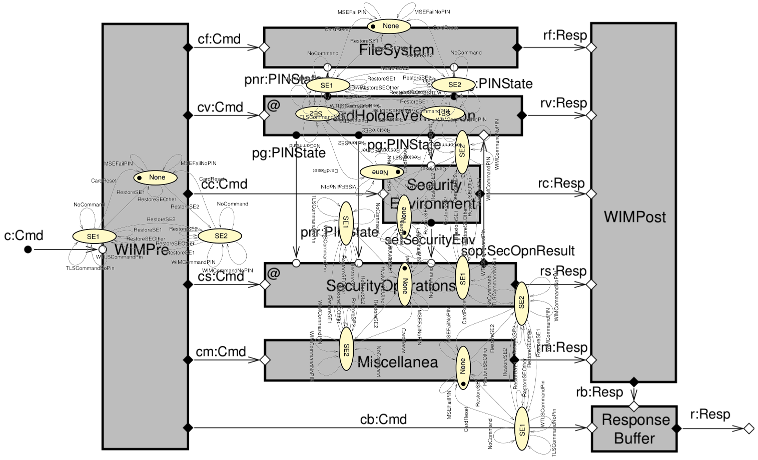
The System to Test



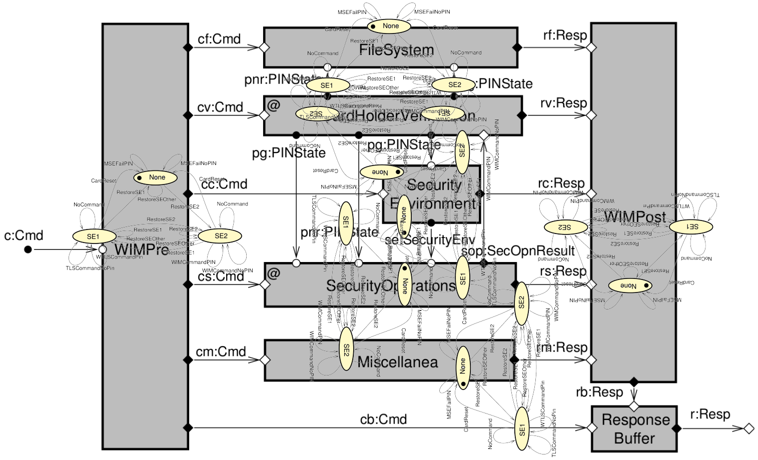
The System to Test



The System to Test

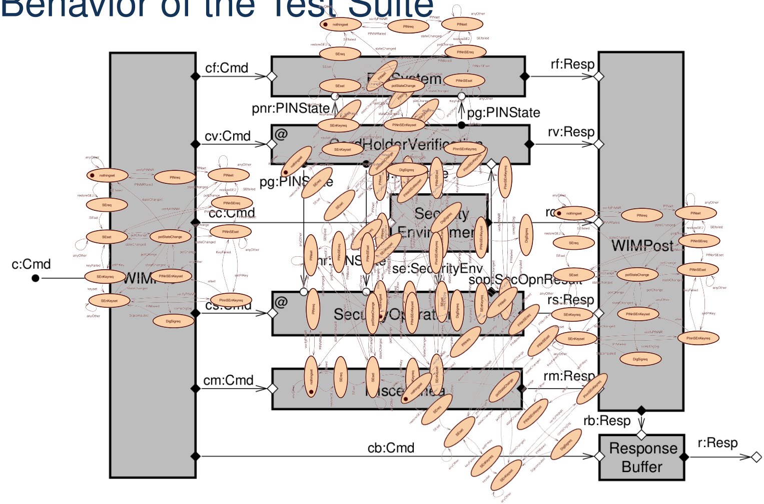


The System to Test



The Behavior of the Test Suite

The Behavior of the Test Suite



Test Coverage and Generation

- Testing cannot (in general) ensure correctness.
- Only existence of errors can be proven.
- Exhaustive testing: very expensive
  (in time to generate tests and to execute them)
- Test generation needs to execute the program.
  - *If I call operation A with data D, which output would I obtain?*

Test Generation for the SmartCard

- Using CLP.
- For a component **a** with a state machine, a collection of clauses like

```
step_a(StateIn, Input, StateOut, Output):-
    guard(StateIn, Input),
    assign(StateIn, Input, StateOut, Output).
```

- Different components can be chained togeter:

```
step_comp(StateIn, In, StateOut, Out):-
    step_a(StateOut, In, S0, O0),
    step_b(S0, O0, S1, O1),
    ...
    step_k(Sn, On, StateOut, Out).
```

- Enumerate traces.
- Use constraints to reduce trace sizes.

CLP to Reduce Traces

- Transition has to read value v from input channel i.

```
v = read(i);
if (v == k) then ...
else ...
```

- In `else` branch: assuming all possible values for `v` not feasible.
- Make a trace with $v = k$ and another with $v \neq k$.
- Constraint in the latter $\rightarrow$ generate refined feasible traces later on.
  - Requiring $v = k$ after that $\rightarrow v \neq k \wedge v = k \rightarrow$ trace not generated.
  - A trace that requires $v \geq k$ is reduced to requiring $v > k$.
- Using sets of values (= constraints) helps reduce the search space.
- Still, some operations are difficult to model.
- `AskRandom(n)`: only length modeled.

Evaluation

- 60.000 test sequences, varying length.
- Testing with only 2%-3% of sequences.
- Around one hour to execute.
- Summary: out of 1506 test sequences, 84 mistmatches.
- All of them due to misinterpretation of documentation or faults in recently optimized versions of software.

Evaluation

- 60.000 test sequences, varying length.
- Testing with only 2%-3% of sequences.
- Around one hour to execute.
- Summary: out of 1506 test sequences, 84 mistmatches.
- All of them due to misinterpretation of documentation or faults in recently optimized versions of software.

1. Make no assumptions.
2. Premature optimization is the root of all evil.

Part V

What Now?

Conclusions

- Yes, formal methods are used in industry — more than usually thought.
- **Not** to write payroll software.
  - But payroll software is already written...
- High-availability, dependable software.
  - NASA; Prolog to formalize JVM class system; SLAM at Microsoft; Esterel for hardware, avionics, and cars; Airbus
    automotive industry, cyber-physical systems.
- Also to discover bugs in existing implementations.
  - E.g., FREAK SSL negotiation bug — team INRIA, Microsoft Research, IMDEA Software Institute.

Recommendations [BH06] and Old Man Sayings

- Use well-tested, well-documented formal method.
- With tool support.
- If possible, have an expert at hand – at least at the beginning.
- Document everything: every assumption, every decision.
- Don’t lower quality standards.
- Test and test again.

Recommendations [BH06] and Old Man Sayings

- Use well-tested, well-documented formal method.
- With tool support.
- If possible, have an expert at hand – at least at the beginning.
- Document everything: every assumption, every decision.
- Don't lower quality standards.
- Test and test again.
- Trying to formalize will force you to think about a problem. Thinking about a problem will make you understand it.

Jean-Raymond Abrial.
The B-Book: Assigning Programs to Meanings.
Cambridge University Press, 1996.

Jean-Raymond Abrial.
Modeling in Event-B: System and Software Engineering.
Cambridge University Press, 2010.

Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier.
METEOR: A Successful Application of B in a Large Project.
In *FM'99—Formal Methods*, pages 369–387. Springer, 1999.

Jonathan Bowen and Michael Hinchey.
Ten Commandments of Formal Methods... Ten Years Later.
Computer, 39(1):40–48, 2006.

Leslie Lamport.
Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.

Addison-Wesley Longman Publishing Co., Inc., 2002.

Chris Newcombe, Rath Tim, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff.
Use of Formal Methods at Amazon Web Services.
2014.

Jan Philipps, Alexander Pretschner, Oscar Slotosch, Ernst Aiglstorfer, Stefan Kriebel, and Kai Scholl.
Model-Based Test Case Generation for Smart Cards.
Electronic Notes in Theoretical Computer Science, 80:170–184, 2003.

Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald.
Formal Methods: Practice and Experience.
ACM Computing Surveys, 2009.

Formal Methods at Work

M. Carro

IMDEA Software Institute
and
Technical University of Madrid

May 11, 2015