New Ways of Using Formal Models in Industry

Michael Leuschel
Overview

• Part 1: Overview of 25 years of history of B
  
  • taken from FMICS 2020 article with Michael Butler, Philipp Körner, Sebastian Krings, Thierry Lecomte, Luis-Fernando Mejia, Laurent Voisin entitled “The First Twenty-Five Years of Industrial Use of the B-Method”

• Part 2: Commandments and Lessons using B and building tools for B, Z, and other formal methods (mainly the tool ProB https://prob.hhu.de) inspired by Bowen, Hall, Hinchey
Part 1: History
Formal Methods

• Mathematical techniques to produce correct software and systems

• Highly recommended e.g. for SIL3/SIL4 railway applications (CENELEC)

• Some Benefits: Problems detected earlier and correction less costly, lower level testing (unit) not required as SW execution errors proven impossible
B Formal Method

Specification

Refinement

Tool Support
Origins of B

- Train protection system SACEM for Paris RER Line A
  - put into operation in 1988, sketch of the B-Method by Jean-Raymond Abrial
- 1989 project by Alstom, RATP, SNCF to develop tools and train engineers
- Paris Metro Line 14 contract won by Matra Transport (now Siemens Transportation Systems)
  - **1995**: B tools industrialised by Digilog (then Steria, now CLEARSY) leading to Atelier-B
  - ready by end 1998: 110 kLOC B model, 83% automatic proof, 86 kLOC Ada
  - Still in version 1.0, no single issue caused by software
B Logical Foundations

- Typed first-order predicate logic with equality
  - Well-Definedness Conditions to stay in two-valued logic
- Arithmetic over mathematical integers and implementable integers
- Set theory
  - Sets, Relations, Functions, Sequences
  - including higher-order functions
- B is simpler than its predecessor Z
  - and provides structuring and refinement for proving and code generation

\[ p \in \text{dom}(a) \implies \text{dom}(a) \wedge \forall i \cdot (i \in 1..(\text{size}(a)-1) \implies p(a(i)) < p(a(i+1))) \]

related state-based formal methods: Z, TLA+, Alloy, VDM, ASM
B Structuring

- Enables **decomposing** a specification
- Ensures that code generated for a B machine can be safely re-used
- Ensures **tractable proof** obligations
- Some key concepts:
  - VARIABLES vs CONSTANTS and associated INVARIANTs and PROPERTIES
  - OPERATIONS to modify variable values
  - Various B machine structuring mechanisms (INCLUDES, USES, SEES, …)
  - REFINEMENT and IMPLEMENTATION machines
**B Structuring**

```
activation_sequence = /* Activation d'une séquence non active */
PRE -(sequences - sequences_actives) THEN
  ANY sequ WHERE
  sequ ∈ sequences - sequences_actives
  THEN
  sequences_actives := sequences_actives ++ \{sequ\}
END.
END:
```

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
  sequ <- indexSequenceInactive;
  activeSequence(sequ)
END;
```

![Diagram of B Structuring process](image-url)
• B for **Software**:

• about 30% of CBTC systems worldwide employ the B formal method

• Urbalis 400, Alstom, over 100 metro lines worldwide, 25% of worldwide CBTC market
“Beyond the technological challenge of using such a complex formal method in an industrial context, it is now clear for us that building software using **B is not more expensive** than using conventional methods. Better, due to our experience in using this method, we can assert that using **B is cheaper when considering the whole development process** (from specification to validation and sometimes certification)”

B for System Modelling
Event-B for System Modelling

• Analyse an entire system of components

• Ensure that together they ensure safety (and functionality)

• Talks about events rather than operations

• Refinement
  • used to structure reasoning, view a system at different levels of granularity
  • requires a more liberal view of refinement
First paper on Event-B already published by Abrial in 1996

Session 6: Chairman, M. Frappier (Univ. de Sherbrooke, Canada)

10h15   Extending B without Changing it (for Developing Distributed Systems)
Invited speaker: Jean-Raymond Abrial (Consultant independent, Paris, F)
Foundations of Event-B

- Better proof automation thanks to simpler substitutions (aka statements) and proof obligations (witnesses, …)
- More expressive and flexible refinement
- Some changes to expressions and predicates
- Foundations realised in the Rodin platform
Tool Support for Event-B

- Rodin
  - Eclipse-based IDE for POG, proof, …
- Atelier B
  - also supports Event-B projects with POG and proof
- ProB
  - Multi-Level Animation, Visualization, Model Checking
CBTC models of ClearSy

• Thales Toronto installed CBTC system for NYC Subway Line 7

• CBTC = Communication-based Train Control: automatic train control system using a combination of classical trackside train detection (TTD) and position reports sent by trains

• ClearSy was asked to perform a formal verification of the safety of the system for Thales Toronto

• from November 2010 until December 2012

• ClearSy was using the Event-B along with the Atelier-B prover

http://www.tools.clearsy.com/resources/formal-proofs-for-the-nyct-line-7-modernization-project/
Summary of Results

- CBTC system **safety** (no collision, no derailment, no overspeeding, correct tracking,…) **formally verified** with B and Atelier-B

- **Key properties** and knowledge extracted and put into the formal model

- Formal **model** was **reused** for NYCT I2S [Sabatier, RSSR16], similar analyses have since then been carried out (Octys CBTC by RATP [Comptier et al., RSSR17],…)

- Showed that a **large** industrial, safety-critical system could be effectively formally analysed and proven correct using Event-B
Alstom Zone Controller

• System analysis carried out in 2018 for a large software system (CBTC Zone Controller) by Alstom, ClearSy and University of Düsseldorf [Comptier et al., RSSR’19]

• Software for this component generated using classical B

• Analysis with Atelier B and ProB

• Enabled to extract **key safety properties** which enable future evolution of the component
ETCS Hybrid Level 3

• Several formal methods case done of the ETCS Hybrid Level 3 (HL3) principles

• B Model and system developed by Thales and University of Düsseldorf

  • Identified over 50 issues in various versions of the HL3 specification

  • Formal model was used in real-time for field demonstration in December 2017 at ETCS National Integration Facility in Hitchin/UK, providing evidence that the HL3 principles are consistent and allow desired operational behaviour
B for Data Validation
Data Validation

• What is the use of a formally proven software if some of its (non trivial) parameters are wrong?

• Data Validation: Automatic check of large data sets against properties

• Properties: international standards, national regulations, manufacturer habits, customer requirements, safety assumptions made during development, …

• E.g. metro line static data used by the automatic pilot (software) to drive safely
B for Data Validation

- Express properties in B: works well with graph-based properties or if software already developed with B

- Initial developments
  - OVADO for RATP, based on predicateB
  - ProB for Siemens in 2008/2009 within Deploy EU project
Aspects of Data Validation

• Focus on expressivity: B language extended (IF-THEN-ELSE, LET for expressions, external functions for string manipulation, regular expressions)

• Tool certification: Tool certified for T2 usage according to EN50128

• extensive testing and validation and/or double chain

• Full automation, scale to large data values, provide user feedback
B for **Data Validation: Industrial Uses**

- Line 1 Paris, the second CDGVAL line LISA at the CDG airport in Paris, São Paulo line 4, ALGER line 1, Barcelona line 9, all by Siemens using **RDV** built-on top of **ProB**,

- more metro lines in Paris managed by RATP using **OVADO** which includes a tool developed called **predicateB** as first chain (development funded by CLEARSY and been maintained and evolved by Systerel for the last 15 years) and **ProB** as secondary tool chain

- Alstom for their URBALIS 400 CBTC system in 2014 using a tool based on **ProB** called **DTVT** developed by CLEARSY for various lines, e.g., in Mexico, Toronto, São Paulo and Panama

- Alstom and SNCF also applied data validation for ETCS-Level 1 software in 2018 using another tool developed by CLEARSY using **ProB**.

- Together with Systerel, Alstom conducted data validation of the Octys CBTC for RATP in 2017 using the **OVADO** tool.

- by Thales using a tool based on ProB called **Rubin** for checking engineering rules of their ETCS Radio Block Centre

- Other tools based on **ProB** were developed by CLEARSY such as **Dave** for General Electric or the latest generation tool called **Caval**.
Reflections

see FMICS’2020 article
The First Twenty-Five Years of Industrial Use of the B-Method

for common success, fail factors,…
Summary: B and its Uses

• B for **software development** (classical B): refine specification until B0, apply code generators
  - Line 14 Paris, Alstom U400, …
  - FM success story, new potential for hardware (LCHIP)

• B for **system modelling** (Event-B): verify critical properties, understand why a system is correct
  - CBTC Flushing Line, NYCT I2S, Octys, Hybrid Level 3, …
  - Activities have increased in last years, potential for executable models

• B for **data validation**: express properties and B and check data (possibly using a double chain)
  - DTVT, Ovado, Dave, Olaf, Caval, Rubin, … for Line 1 Paris, Amsterdam, …
  - FM success story, widespread usage in railway industry
Part II: Commandments and Lessons

I. Thou Shalt

for a) using B and

for b) building tools for B, Z, and other formal methods
Inspiration


Thou Shalt Animate your Models

• ensures your assumptions are consistent (there is at least one model)

• allows to spot errors which are impossible to avoid using invariants and difficult to describe using temporal logic

• spots class of errors you haven’t thought of yet
“Every formal model (proven or not) which has not been animated contained errors”

Christophe Metayer, Systerel

(liberal translation from French based on verbal communication)
Earley Parsing Algorithm Model

Fully proven, but ProB found inconsistency in the axioms (->->> instead of ->->) during animation.
Thou Shalt Visualize
\[ \text{vss\_left} = \{ \ldots , "17VTS33" \rightarrow 140, "17VTS34" \rightarrow 240, "17VTS34" \rightarrow 340, \ldots \} \]
\[ \text{TTD\_state} = \{ \text{TTD1} \rightarrow \text{free}, \text{TTD2} \rightarrow \text{occupied}, \text{TTD3} \rightarrow \text{occupied}, \ldots \} \]
\[ \text{vss\_state} = \{ \ldots , "17VTS33" \rightarrow \text{unknown}, "17VTS34" \rightarrow \text{unknown}, "17VTS35" \rightarrow \text{occupied}, \ldots \} \]

\[ \text{Env\_train\_length} = \{ \text{train1} \rightarrow 30, \text{train2} \rightarrow 30 \} \]
\[ \text{Env\_train\_FP} = \{ \text{train1} \rightarrow 250, \text{train2} \rightarrow 350 \} \]

\[ \text{registeredTrains} = \{ \text{train2} \} \]
\[ \text{train\_reported\_integrity} = \{ \text{train2} \rightarrow \text{confirmed\_integrity} \} \]
ProB2-UI Demo

Operations View for interactive animation

State View to inspect current and preceding state

Project View for models and preferences

Replay View for automatic trace replay

Console (REPL) for interactive exploration

History View to inspect and navigating current animation trace

VisB View SVG-based visualization of current state

https://prob.hhu.de
Simple Train Model

- Track is an interval $0..\text{TrackElementNumber}$
- Track is divided into TTD (Trackside Train Detection) zones (e.g. implemented using track circuits or axle counters)
- Trains have positions $\text{train\_rear\_end}(\text{tr})..\text{train\_front\_end}(\text{tr})$
- Some trains have an MA (Movement Authority) extending beyond their front end
- Many things not modelled: delays, position reports, uncertainty of train image, train speed, braking curves, points, train integrity, ...
Some Points

- Animation and Visualisation help me understand models others have written

- also help make your model better understandable to other people, even domain experts not able to read your formal notation
Formal mathematics is nature's way of letting you know how sloppy your mathematics is.

Animation and Visualization is nature's way of letting you know how sloppy your formal mathematics is.
Thou Shalt Not Abandon Thy Traditional Formal Proof Methods

• alternatively:
  Thou Shalt Use the Trinity of Methods
The Trinity of Methods

• I tend to use proof, model checking and animation together

• **Proof**: solves the state explosion problem, provides key (inductive) properties and insights

• **Model checking**: finds obvious problems, increases confidence that proof is feasible, checks liveness properties

• **Animation**: validate scenarios (often part of the requirements), find inconsistencies, detect “surprising” and obvious errors quickly
Conflict of Interest

• Proof and animation have conflicting needs:

• adding an axiomatic property for proof can make finding a valid model much harder

\[ \forall s \in \text{Track} \Rightarrow P(s) \]

• adding concrete data and constructive definitions for animation can make proof harder and less general

\[ f = \lambda x. \text{x}\in \text{Train} | \text{front(x)..<train_ma(x)} \]

• Use refinement to create model checking and animation instances

• Annotate/isolate complex properties (@prob-ignore pragma)
The diagram illustrates a "Classical" Refinement Approach with a flowchart detailing the processes:

1. **M0** to **M1** - Proof
2. **M1** to **M2** refinement
3. **M2_a** Instantiation for Validation
4. **M2_b** Instantiation for Validation
5. **Impl** - Lowest Level
6. **Generate Statespace MC2**
7. **Human Check of Abstract Statespace H1**
8. **Use Case Replay U1**
9. **Temporal Property MC1**
10. **Model Checking**
11. **Animation**
Thou Shalt Reuse
Thou Shalt Reuse

💡 Ideas, not Models
Thou Shalt Reuse Ideas, not Models

• When modelling:
  
  • Good idea to reuse: key concepts, ways to decompose a system, approach to ensure inductive proof is possible
  
  • Usually difficult to reuse formal models for modelling
Thou Shalt Use Models as Documentation

- Static Documentation
- Executable, interactive Documentation (HL3)
Dear Thomas N. Ginn,

Please find attached a scenario of our formal model. Can you inspect whether the behaviour corresponds with your expectations. You can just open the file in any browser and replay and inspect the trace.

Kind regards,
Michael

TwoTrainsMA.html

Open
Dear Thomas N. Ginn,

Please find attached a scenario of our formal model. Can you inspect whether the behaviour corresponds with your requirement? You can just open the file in any browser and replay and analyze the scenario.

Kind regards,
Michael

TwoTrainsMA.html (16.1 KB)
ProB Jupyter Notebooks for Documentation

**KISS PASSION Puzzle**

A slightly more complicated puzzle (involving multiplication) is the KISS * KISS = PASSION problem.

```python
In [3]:
1   {K,P} \subseteq 1..9  
2   {I,S,A,0,N} \subseteq 0..9  
3   (10000*K+100*I+10*S+S) *  
4       (1000*K+100*I+10*S+S)  
5       = 1000000*K+100000*I+10000*S+S+10000*I+100*S+10*I+O+N  
6   \text{card}({K, I, S, P, A, 0, N}) = 7

Out[3]: \text{TRUE}
```

**Solution:**

- \( P = 4 \)
- \( A = 1 \)
- \( S = 3 \)
- \( I = 0 \)
- \( K = 2 \)
- \( N = 9 \)
- \( O = 8 \)
ProB Jupyter Notebooks for Documentation

Mixing Functional Programming with Constraint Programming

In B we can also define (higher-order) functions and mix those with logical predicates.

In [12]:

\[ f = \lambda x. (x \in \mathbb{Z} \land x \times x) \land \text{res}=\{x \mid f(x) = 100\} \]

Out[12]: \( \text{TRUE} \)

Solution:

- \( \text{res} = \{-10, 10\} \)
- \( f = \lambda x. (x \in \text{INTEGER} \mid x \times x) \)
Thou Shalt Use Execute your Models

- Oracle in Test Environment (Advance)
- Executable Prototype for early field tests (HL3)
- Long term: maybe formal models in the loop in the final product (Plues tool)
Project for ProRail

- Field demonstration of the ETCS Hybrid Level 3 (HL3) principles

- Demonstration by co-operation trackside (Thales, Siemens) and onboard (Alstom, Hitachi)

- Demonstration line: ETCS National Integration Facility (ENIF) in Hitchin/UK

- There was insufficient time to model and code a prototype

- But there was sufficient time to embed the formal model at runtime
Natural Language Specification

New System
New Feature

Formal Model

Testing, Debugging

Execution in real environment
ProRail Project Summary

• using model as demonstrator/prototype is feasible, there were a lot of technical issues, ProB was not one of them!

• animation/visualization can help understand and debug a given specification

• using a formal model allowed to quickly adapt the model as fixes for issues came along, several new requirements were integrated

• log of formal model could be replayed step-by-step to analyse issues
The use of formal methods in specification and demonstration of ERTMS Hybrid Level 3

Prepared on behalf of the International Technical Committee by Maarten Bartholomeus, Bas Luttik, Tim Willemsen, Dominik Hansen, Michael Leuschel and Paul Hendriks
ProB in Action: Formal Models in Realtime

ProB running in real-time animating a formal B model of the Hybrid-Level 3 principles developed by a team from the University of Düsseldorf and Thales with support from ClearSy.

Train 2 following Train 1 (Lucy) on the same occupied track section but on different virtual subsections.

https://www.youtube.com/watch?v=FjKnugbmrP4
from modelling to tools
Thou Shalt Choose an Appropriate Notation/Programming Language for your Tool

- Prolog for type checking, rule-based theorem proving, constraint solving
- Java, Tcl/Tk, … for user interface
- C for LTL model checking
- …
The heart of ProB is written in Prolog, but other languages are used around it:

- Java
- C, C++
- Tcl/Tk
- Haskell
Hindley-Milner Type Inference

• Easy to encode in Prolog: one type inference rule is one Prolog clause

• More powerful than Atelier-B, ...

• cf. VPT-2020 article

• Fast

```prolog
// type([],set(_)) --> !, []).  
// type(union(A,B),set(R)) --> !,type(A,set(R)), type(B,set(R)).  
// type(intersect(A,B),set(R)) --> !,type(A,set(R)), type(B,set(R)).  
// type(plus(A,B),integer) --> !,type(A,integer), type(B,integer).  
// type(in_set(A,B),predicate) --> !,type(A,TA), type(B,set(TA)).  
// type(gt(A,B),predicate) --> !,type(A,integer), type(B,integer).  
// type(and(A,B),predicate) --> !,type(A,predicate),type(B,predicate).  
// type(eq(A,B),predicate) --> !,type(A,TA),type(B,TA).  
// type(Nr,integer) --> {number(Nr)},!.  
// type([H|T],set(TH)) --! type(H,TH), type(T,set(TH)).  
// type(ID,TID) --> {identifier(ID)},\+ defined(id(ID,_)),!,  
// add((id(ID,TID))). % creates fresh variable  
// type(ID,TID) --> {identifier(ID)},defined(id(ID,TID)),!.  
// type(Expr,T,Env,_):=  
// format('Type error for ~w (expected: ~w, Env: ~w)~n', [Expr,T,Env]),fail.  
// defined(X,Env,Env):- member(X,Env).  
// add(X,Env,[X|Env]).  
// identifier(ID):- atom(ID), ID \= [].  
// type(Expr,Result):- type(Expr,Result,[],Env),format('Typing env: ~w~n',[Env]).
```

```
{z} \cup \{x,y\} = u \land z > v
```

| %- type(and(eq(union([z],[x,y]),u),gt(z,v)),R).  Typing env: [id(v,integer),id(u,set(integer)),id(y,integer),id(x,integer),id(z,integer)]  
R = predicate ?  yes |
Anecdotal Evidence:
Typechecking 8000 Line B specification
Semantic Translation Rules: Alloy2B

- Translator of Alloy [Jackson] to B
- Adaptation of formal semantics of Alloy by simply using B syntax
- Rules can be translated to Prolog clauses
- First version was written in Kotlin (JVM), then switched to Prolog as error prone and tedious to encode rules

\[
E[p + q] \equiv E[p] \cup E[q] \\
E[p \& q] \equiv E[p] \cap E[q] \\
E[p - q] \equiv E[p] \setminus E[q]
\]

```
translate_binary_e_p(Binary, TBinary) :-
    Binary =.. [Op,P,Q,_,POS],
    alloy_to_b_binary_operator(Op, BOp),
    translate_e_p(P, TP),
    translate_e_p(Q, TQ),
    translate_pos(POS, BPOS),
    TBinary =.. [BOp,BPOS,TP,TQ].

alloy_to_b_binary_operator(plus, union).
alloy_to_b_binary_operator(intersection, intersection).
alloy_to_b_binary_operator(minus, set_subtraction).
alloy_to_b_binary_operator(implication, implication).
alloy_to_b_binary_operator(iff, equivalence).
...```


Prolog Theorem Prover for Proving Well-Definedness

- WD Prover [iFM’2020] to prove absence of division by zero, undefined function applications, cardinality of infinite sets, …

- Shared Hypothesis stack
  - `pop` via **Prolog backtracking**
  - Only **logarithmic** accesses to Hypotheses

- Efficient **rule-based prover** using **Prolog unification**
The range of a function is finite if the function is finite.

In Java:
82 lines of code
(9 lines are copyright notice)

The Prolog code is also very flexible: it can be used for finding proofs but also for re-playing or checking proofs if the proof tree argument is provided.
public class FiniteRan extends EmptyInputReasoner {

    @Override
    public String getReasonerID() {
        return REASONER_ID;
    }

    @ProverRule("FIN_REL_RAN_R")
    protected IAntecedent[] getAntecedents(IProverSequent seq) {
        Predicate goal = seq.goal();
        // goal should have the form finite(ran(r))
        if (!Lib.isFinite(goal))
            return null;
        SimplePredicate sPred = (SimplePredicate) goal;
        if (!Lib.isRan(sPred.getExpression()))
            return null;
        // There will be 1 antecedents
        IAntecedent[] antecedents = new IAntecedent[1];
        UnaryExpression expression = (UnaryExpression) sPred.getExpression();
        Expression r = expression.getChild();
        final FormulaFactory ff = seq.getFormulaFactory();
        // finite(r)
        Predicate newGoal = ff.makeSimplePredicate(Predicate.KFINITE, r, null);
        antecedents[0] = ProverFactory.makeAntecedent(newGoal);
        return antecedents;
    }
}

Prolog

check_finite(range(A), Hyp, ran(PT)) :- !,
    check_finite(A, Hyp, PT)

Java
**Biased Benchmarks**

- Test Atelier-B provers ML, PP and Z3 on 413 POs from ProB regression tests

- **Biased**, but shows that our prover is **fast** and can prove POs not proven by existing provers (in default settings)

<table>
<thead>
<tr>
<th>Prover</th>
<th>Proved</th>
<th>Unproved</th>
<th>Ctrl-C</th>
<th>Min.</th>
<th>Max.</th>
<th>Total</th>
<th>w/o Min.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProB-WD</td>
<td>413</td>
<td>0</td>
<td>0</td>
<td>0.000 sec</td>
<td>0.006 sec</td>
<td>0.047 sec</td>
<td>0.047 sec</td>
</tr>
<tr>
<td>ML</td>
<td>190</td>
<td>223</td>
<td>0</td>
<td>0.260 sec</td>
<td>18.725 sec</td>
<td>152.633 sec</td>
<td>45.253 sec</td>
</tr>
<tr>
<td>PP</td>
<td>230</td>
<td>165</td>
<td>18</td>
<td>0.092 sec</td>
<td>49.144 sec</td>
<td>222.940 sec</td>
<td>186.600 sec</td>
</tr>
<tr>
<td>Z3</td>
<td>9</td>
<td>14</td>
<td>0</td>
<td>0.007 sec</td>
<td>2.520 sec</td>
<td>1017.406 sec</td>
<td>-</td>
</tr>
</tbody>
</table>
A few more commandments

• Thou Shalt Honour Text and Command-Line Interfaces:
  • Text and command-line tools are not dead yet

• Thou Shalt Think Twice before Redeveloping your Tool from Scratch
  • Redeveloping a tool or even a UI from scratch is hard and will take longer than you think

• Thou Shalt Obtain User Feedback and User Your Own Tool
  • make it easy for users to provide feedback; use your tool yourself
Summary: Thou Shalt

1. Animate Your Models
2. Visualize Your Models
3. Reuse Ideas, not Models
4. Not Abandon Thy Traditional Formal Proof Methods
5. Use Models as Documentation
6. Execute Your Models
7. Choose an Appropriate Notation/Programming Language for Your Tool
8. Honour Text and Command-Line Interfaces
9. Think Twice before Redeveloping your Tool from Scratch
10. Obtain User Feedback and User Your Own Tool
Outlook: New Projects

- IVOIRE (with Univ. Linz)
  - Automate Validation in a Refinement-based Development Process
- KI-LOK
  - Certifying Railway Systems with AI
Formal Methods in Industry

- Formal methods in general and B in particular can be used to verify/validate:
  - **code** of individual components
  - **systems** consisting of components (algorithms, design,...)
  - **configurations** of components

- Formal methods have become much more useful thanks to **progress** in proof, constraint solving (SAT, SMT, CLP), visualization
  - tools can find errors which no human could ever find
  - tools can guarantee the absence of certain errors
  - tools can help visualize and understand very complex systems, behaviours and interactions
Thank you very much

Thanks to

Jens Bendisposto
Carl Friedrich Bolz
Michael Butler
Joy Clark
Ivo Dobrikov
Jannik Dunkelau
Nadine Elbeshausen
Fabian Fritz
Marc Fontaine
Marc Frappier
David Gelessus
Stefan Hallerstede
Dominik Hansen
Christoph Heinzen
Michael Jastram
Philipp Körner
Sebastian Krings
Lukas Ladenberger
Li Luo
Thierry Massart
Daniel Plagge

Antonia Pütz
Mireille Samia
Joshua Schmidt
David Schneider
Corinna Spermann
Sebastian Stock
Yumiko Takahashi
Edd Turner
Michelle Werth
Dennis Winter
Fabian Vu

Aistom (Fernando Mejia, …)
ClearSy (Thierry Lecomte, …)
Siemens
Systerel
Thales (Nader Nayeri, Georg Hemzal, …)

Danny De Schreye
Maurice Bruynooghe
Bart Demeun
Marc Denecker
Bern Martens
Wim Vanhoof

Robert Glück
Neil D. Jones
Jesper Jørgensen
Torben Møgensen

Fabio Fioravanti
Alberto Pettorossi
Maurizio Proietti
Emanuele De Angelis

John Gallagher
Manual Hermenegildo
German Puebla
Josep Silva
Salvador Tamarit
German Vidal

Kostis Sagonas
and many more

Thank you
very much