History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

BRILLANT: An Open Source and XML-based platform for Rigourous Software Development

Samuel Colin[1] Dorian Petit[1] Jérôme Rocheteau[2]
Rafaël Marcano[2] Georges Mariano[2] Vincent Poirriez[1]

[1]LAMIH/ROI, UMR CNRS 8530
University of Valenciennes

[2]INRETS/ESTAS
INRETS, institute of Villeneuve d'Ascq

Software Engineering and Formal Methods, 2005

Outline

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

1 History
2 The core tools (BCaml)
3 UML/B
4 BPhoX
5 Code generation
6 Conclusion & perspectives

The need for tools
The context

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

1991	Idea of the B method
1996	B-Book [Abrial, 1996]
1998	Meteor
...	...

Circa 2000:

- Only tools available: `AtelierB` (Steria – now Clearsy), `B Toolkit` (BCore)
- A formal method can never be safe enough:
  - Corrections, or extensions to the B method arise
  - Tools above not suitable to experiment said extensions

$\Rightarrow$ Problems related with the “black box”-type approaches

The need for tools
The choices

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- INRETS: French national institute for transport and safety research. B is used for designing (a part of) *Meteor*, an automated subway train
- How safe should the tools for using a formal method be ?
  - *Quis custodiet ipsos custodies*? (Who shall keep the keepers themselves?)
- Free, open-source steps for developing an expandable platform for a formal method:
  - Science is about exchanging ideas
  - Formal methods are about developing in a most rigorous way

From the parser to the platform

Initial choices

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

We first needed to be able to *simply* parse B specifications.

- A well-established parsing technology
- A language to manipulate abstract data structures
- The language had to be based on technologies as safe and as usable as possible
- Ideally, other treatments for the B abstract structures had to be achievable without too much work.

From the parser to the platform
The growth

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

Once the abstract structure of a B project is at disposal, we can manipulate it at will:

- Generate proof obligations of a B project
- Prove these proof obligations
- Generate code
- Use the tools as a *plug-out*. B becomes the target of another tool, such as a convertor of UML specifications.
- Add support for ill-defined expressions, increase the usability of the tool,. . .

$\Rightarrow$ BRILLANT: “B: research and software innovations thanks to new technologies”.

Outline

1 History
2 The core tools (BCaml)
3 UML/B
4 BPhoX
5 Code generation
6 Conclusion & perspectives

Parsing/printing technology

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

Chosen programming language: Objective Caml

- Based on decades-long research in typing and functional programming $\Rightarrow$ *acceptable safety level*
- Numerous libraries for the manipulation of abstract data structures are available $\Rightarrow$ *pretty usable*
- Libraries for interfacing OCaml and/or said structures to other tools $\Rightarrow$ *extendable*
- Can generate native-code, and (faster than java) byte-code executables $\Rightarrow$ *Efficient*
- Clear correspondence between the theoretical rules and the tools supporting it

Parsing/printing technology
(continued)

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- Parsing technology: (Ocaml)Lex+(OCaml)Yacc.
  - Long established, well understood, formal technology
  - Helped highlight ambiguities in the B grammar
  - Led to the publishing of the first mechanized grammar covering the whole B language (records, definitions,. . .)
- Chosen exchange format: XML
  - Collaborative development
  - *Only* an exchange format: no more, no less.
  - Represents unambiguously B abstract syntax
  - Simple structure checks (DTDs, schemas)

Abstract manipulations

Modularity of B projects

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- Flattening a B project:
  - Building a single B machine equivalent to a whole project (useful for generating code)
  - “As-is” implementation of an existing algorithm
  - Relies on functions manipulating dependency graphs
- Alternative semantics of the modularity of B: B-HLL
  - Inspired by works in the functional programming world
  - The modular links of B are given a much simpler semantics

Abstract manipulations
Generation of proof obligations

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- Following the B-Book “by the book”:
  - Define a simpler substitution language (*Generalised Substitutions Language* – GSL for short)
  - Implement the weakest precondition calculus (WPC)
  - Implement the functions for generating the different classes of proof obligations
- But we can do more:
  - Generate the formulas *without* applying the WPC
  - Optimize the shape of the obtained formulas
  - Embed trace information into the proof obligations (“where does this variable come from ?”).
- Still using an XML format

Outline

1 History
2 The core tools (BCaml)
3 UML/B
4 BPhoX
5 Code generation
6 Conclusion & perspectives

Formalizing UML models

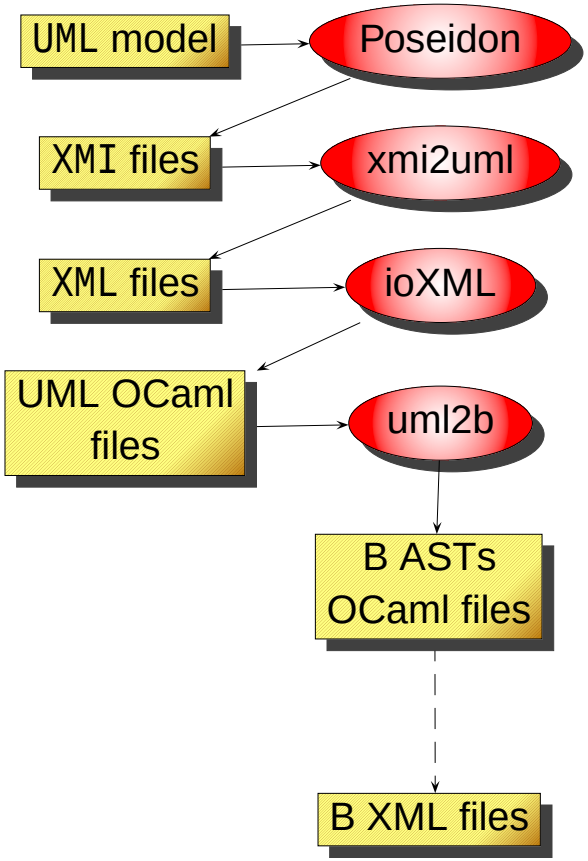
History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

Goal: to be able to validate UML models. These are translated into B to do so (see [Marcano and Levy, 2002]):

- Classes $\equiv$ Sets representing instances of the classes
- Relations between classes $\equiv$ set-theoretic relations
- States $\equiv$ operations of the machine
- OCL constraints:
  - Invariants become part of the machine's invariant
  - Preconditions become part of operations' preconditions
  - Postconditions become part of operations' substitutions

Mechanisms of the tool

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



Tightly connected to *BCaml*

Generation of XML files still possible

Outline

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

1 History
2 The core tools (BCaml)
3 UML/B
4 BPhoX
5 Code generation
6 Conclusion & perspectives

A prover for B

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- Developing another prover for set-theoretic predicate calculus is not worth the effort:
  - Takes a lot of time and manpower
  - Such provers already exist
- Possible ideas:
  - To adapt an existing prover $\Rightarrow$ compatibility between B-Book rules and the rules of the prover
  - To use a more “general-purpose” prover (PVS, Isabelle/HOL, Coq) $\Rightarrow$ write the missing libraries

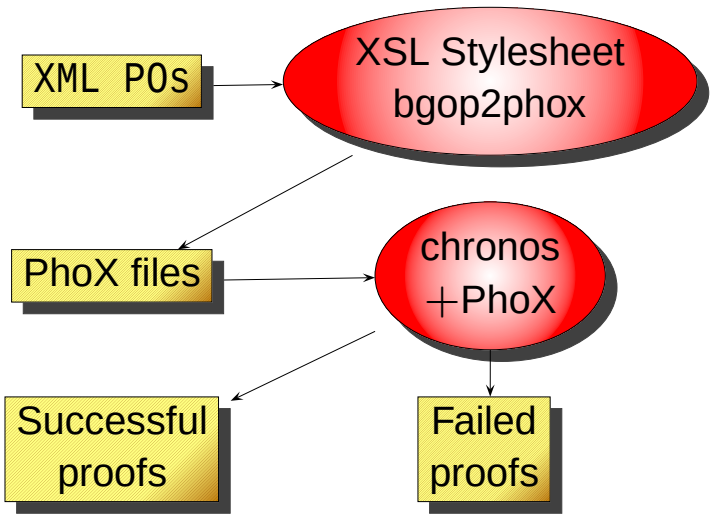
The chosen One

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- We chose PhoX: Why (see [Rocheteau et al., 2004]) ?
  - Its GPL license (it thus can be redistributed with other GPL tools such as BCaml)
  - Its developers desire to work closely with us
  - As intuitive to use as possible
  - Developers of the B libraries are comfortable with higher-order reasoning
- *Coq* could be another candidate

Mechanisms of the tool

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



Example of results:

BOILER project, 1s timeout:
79% success

BOILER project, 5s timeout:
85% success

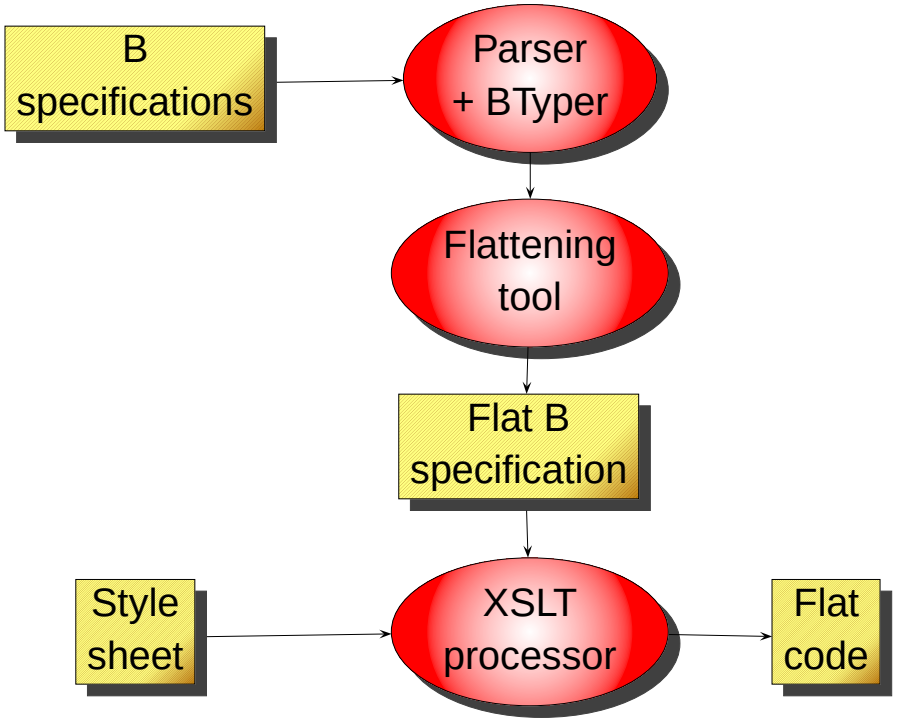
BOILER project, 60s timeout:
85% success

Outline

1 History
2 The core tools (BCaml)
3 UML/B
4 BPhoX
5 Code generation
6 Conclusion & perspectives

Flat code generation

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

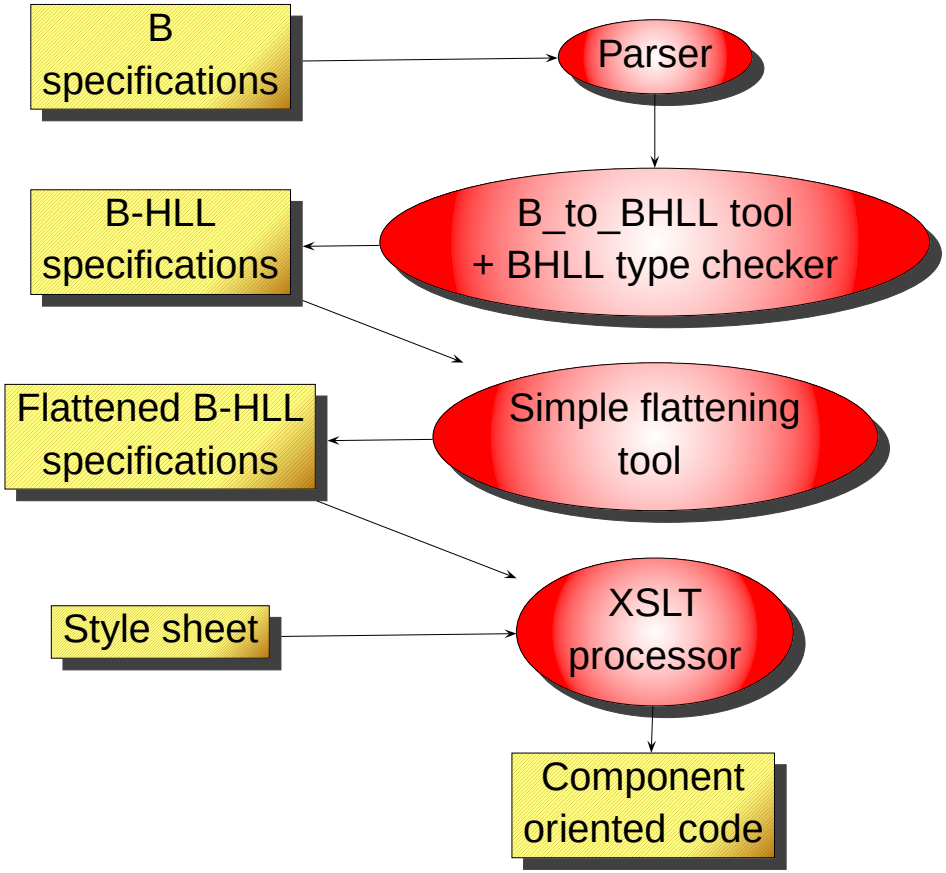


A typing phase is necessary (typing information can not be extracted from invariants).

There is one stylesheet per target language.

Component-oriented code generation

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



For keeping modular information in the target source code.

Contracts (invariants, pre-conditions of B machines) can be preserved.

See [Petit et al., 2004].

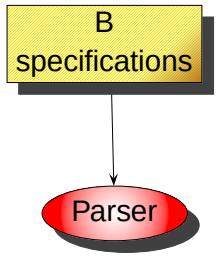
Outline

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

1 History
2 The core tools (BCaml)
3 UML/B
4 BPhoX
5 Code generation
6 Conclusion & perspectives

The big picture

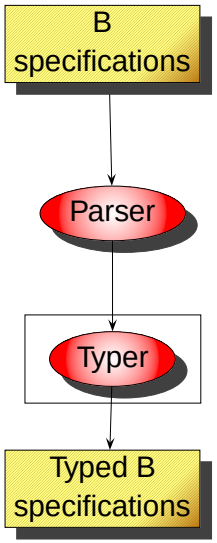
History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



IDExp	=	Ill-defined expressions
XSLTProc	=	XSLT processor
PhoX	=	Proof assistant
POG	=	Proof obligations (POs) generator

The big picture

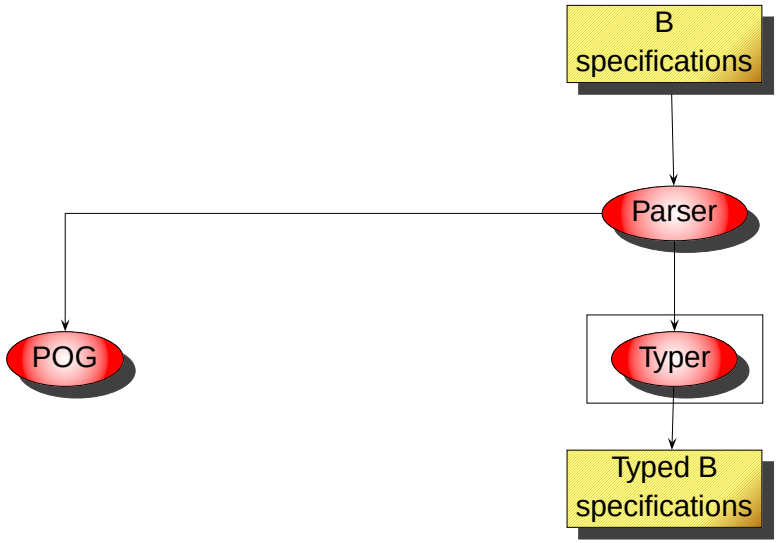
History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



IDExp	=	Ill-defined expressions
XSLTProc	=	XSLT processor
PhoX	=	Proof assistant
POG	=	Proof obligations (POs) generator

The big picture

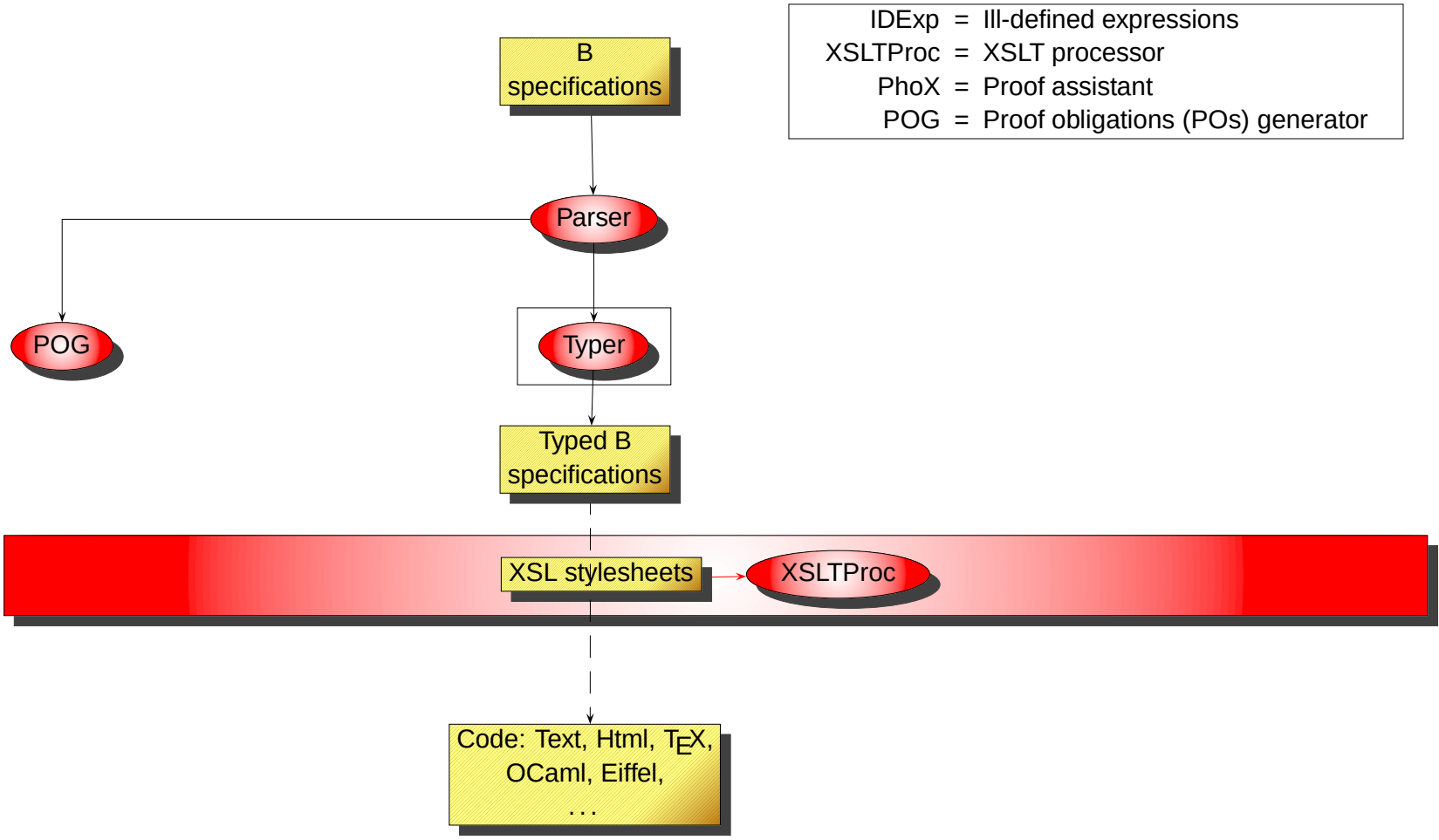
History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



IDExp	=	Ill-defined expressions
XSLTProc	=	XSLT processor
PhoX	=	Proof assistant
POG	=	Proof obligations (POs) generator

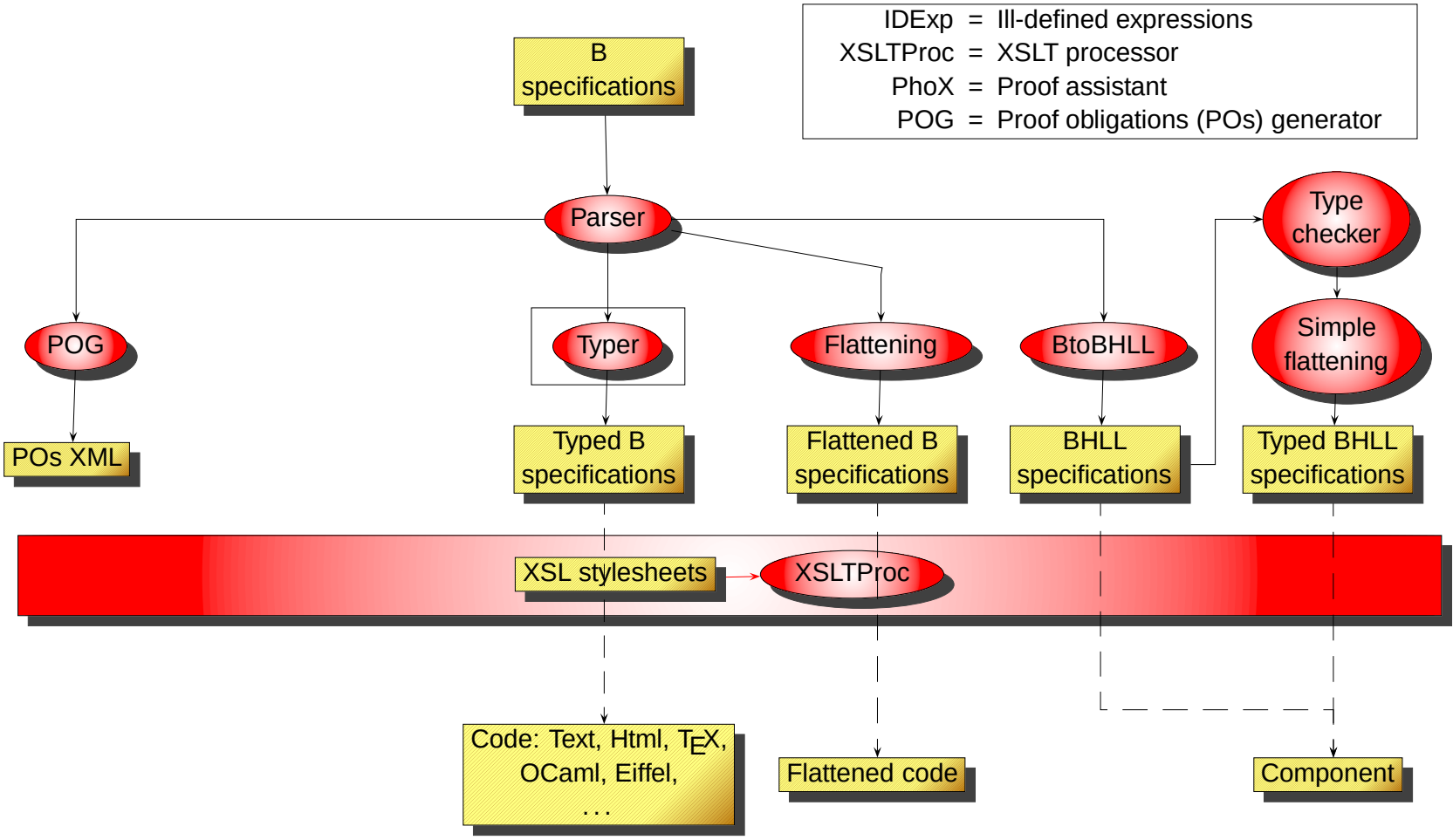
The big picture

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



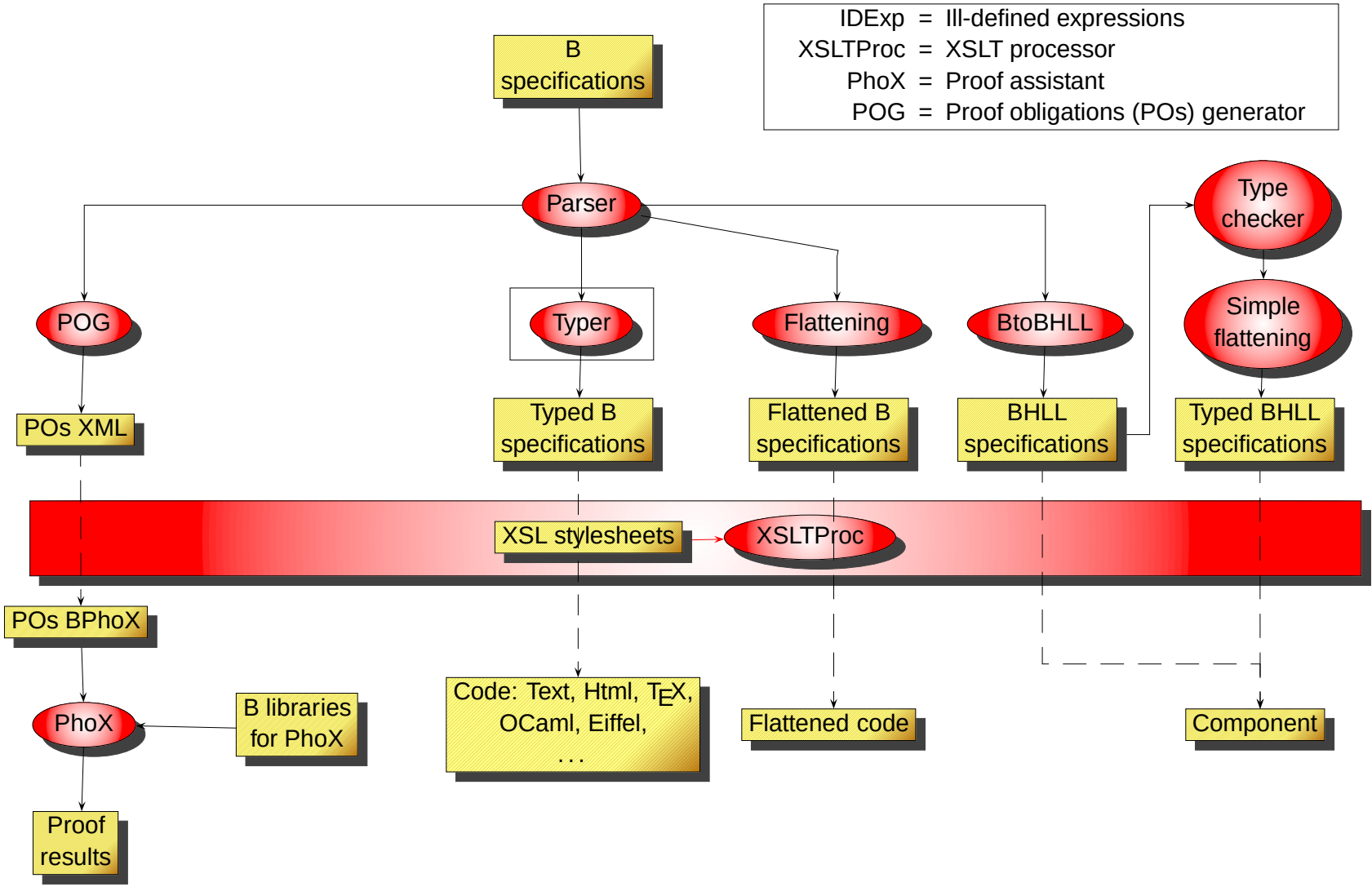
The big picture

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



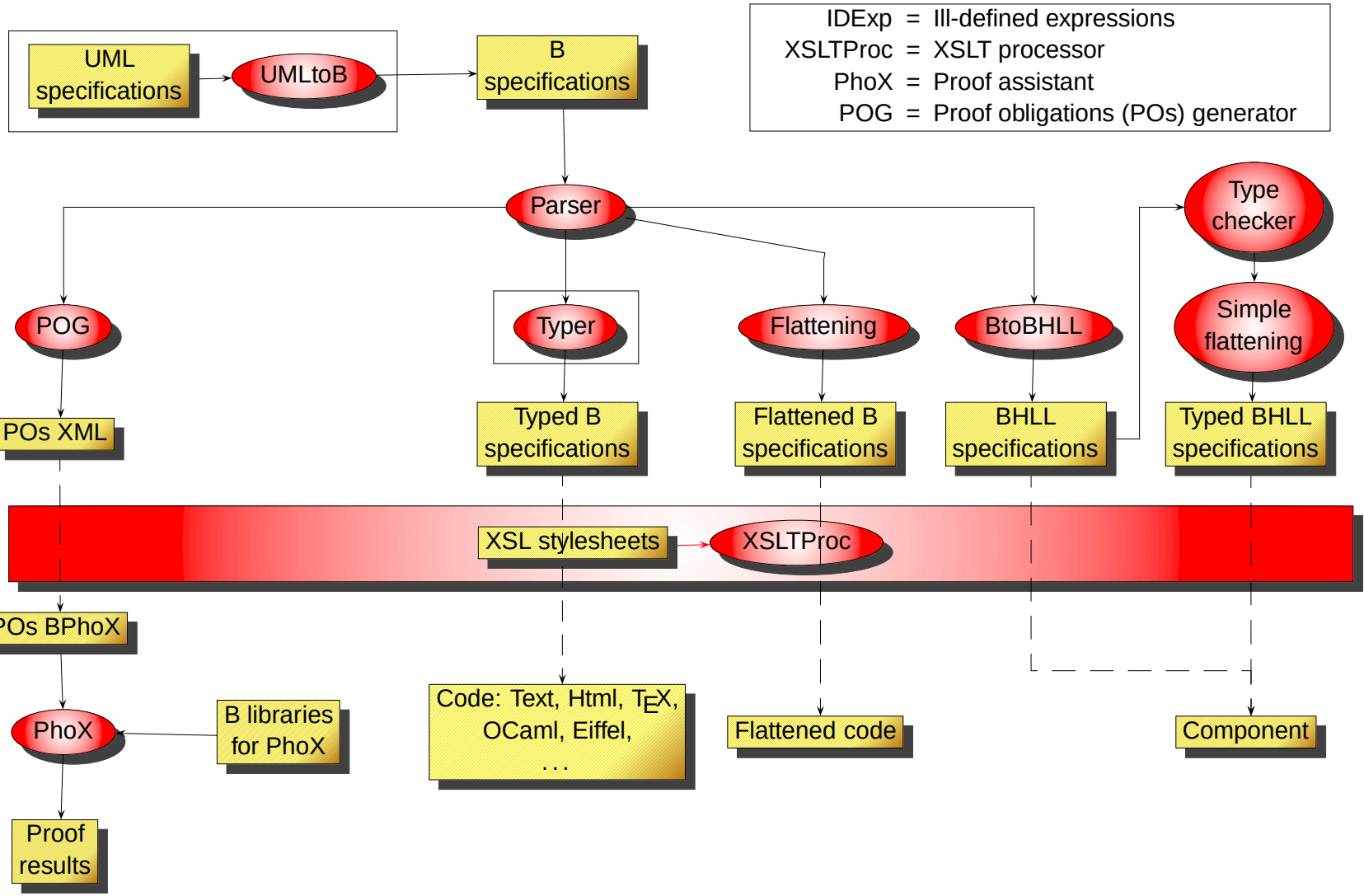
The big picture

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



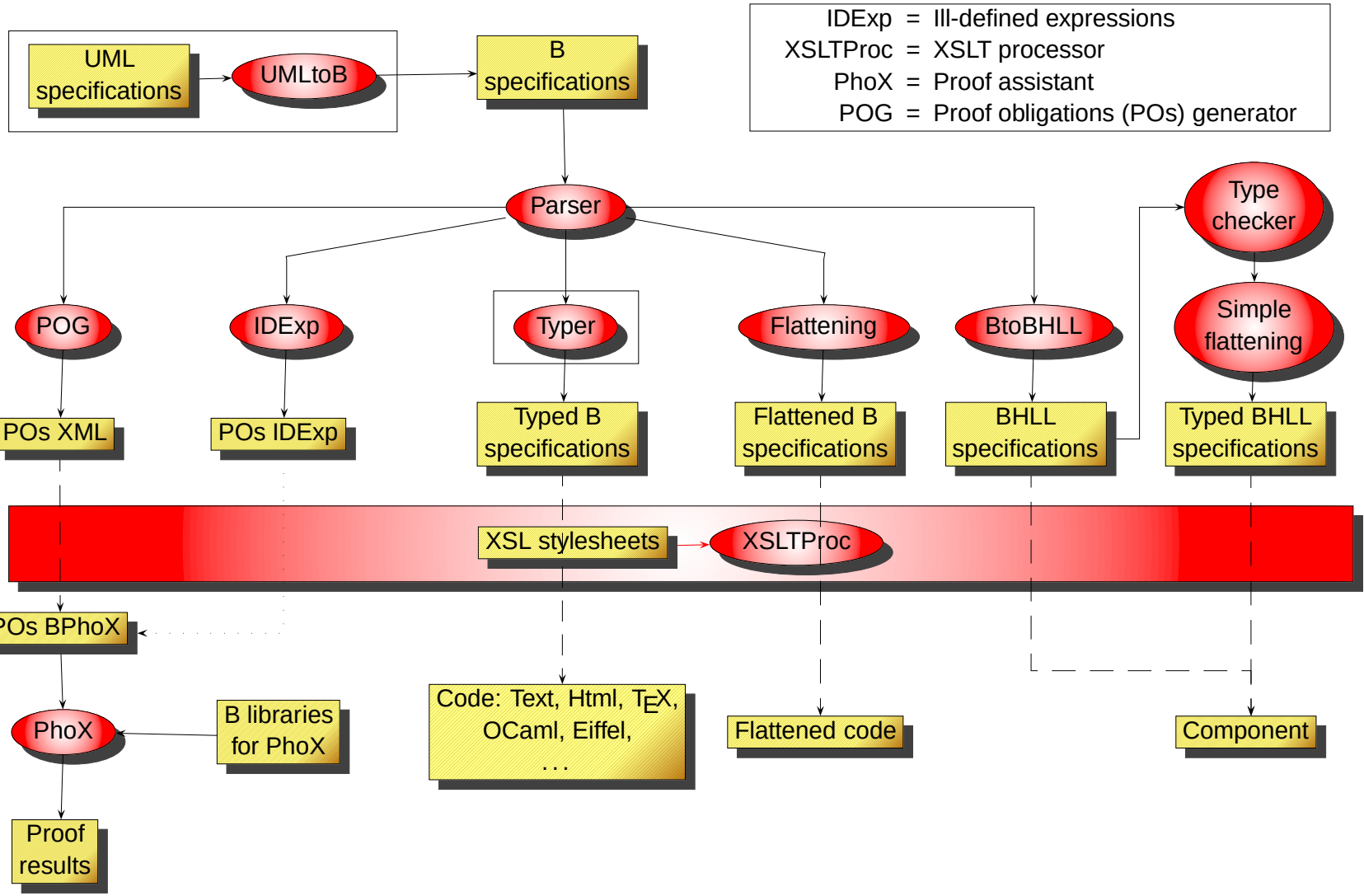
The big picture

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



The big picture

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions



Thoughts so far

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- What characteristics are desirable for a rigorous software development ?
  - High level: either the language (OCaml) or the methods (ANTLR)

- The necessity of these characteristics is enforced by the development context (Master students, PhD students)

Thoughts so far

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- What characteristics are desirable for a rigorous software development ?
  - High level: either the language (OCaml) or the methods (ANTLR)
  - Suited: the task of each separate component suits the chosen tool (OCaml, PhoX, XML as an exchange format)

- The necessity of these characteristics is enforced by the development context (Master students, PhD students)

Thoughts so far

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- What characteristics are desirable for a rigorous software development ?
  - High level: either the language (OCaml) or the methods (ANTLR)
  - Suited: the task of each separate component suits the chosen tool (OCaml, PhoX, XML as an exchange format)
  - Simple: the tools shall allow simpler development or simpler use (PhoX usability, UML for model development)

- The necessity of these characteristics is enforced by the development context (Master students, PhD students)

Thoughts so far

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- What characteristics are desirable for a rigorous software development ?
  - High level: either the language (OCaml) or the methods (ANTLR)
  - Suited: the task of each separate component suits the chosen tool (OCaml, PhoX, XML as an exchange format)
  - Simple: the tools shall allow simpler development or simpler use (PhoX usability, UML for model development)
  - Automatic: where possible, process automatically data or code snippets (XSL stylesheets for code generation)
- The necessity of these characteristics is enforced by the development context (Master students, PhD students)

Comparison with other tools

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

Let us try to give characteristics of other similar tools

- Rodin (event B).
  - Eclipse IDE (simple, automatic, suited)
  - Appeared 2004-05, event B specs published 2005-07
- ProB (B method)
  - Animation, model-checking, Prolog, XML (high level, suited, automatic)
  - Appeared 2003-07, first alpha release 2003-10, 1.0.0 release 2004-03
- CZT (Community Z Tools)
  - Similar with BRILLANT. Java, jEdit, Z extensions (suited, simple)
  - Appeared 2001-07, first beta release 2004-12

Other tools
A quick glimpse

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- ABTools (B method): programmed in Java, uses ANTLR, is part of BRILLANT
- Ebba (B method): corrects some B typechecking rules, programmed in Haskell
- Hets (CASL): based on algebraic specifications, programmed in Haskell
- Many others...

Undergoing and future development

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions

- Treatments of ill-defined expressions in B
- Ergonomic interaction mode (again, based on the XML format)
- Move towards a communicating framework (XML-RPC)
- B models as projects databases (XPath, XQuery)
- Temporal extension of the B method [Colin et al., 2004]
- B event ?

For Further Reading I

History
The core tools (BCaml)
UML/B
BPhoX
Code generation
Conclusion & perspectives
References
Questions


Abrial, J.-R. (1996).
The B Book - Assigning Programs to Meanings.
Cambridge University Press.


Colin, S., Mariano, G., and Poirriez, V. (2004).
Duration calculus: A real-time semantic for B.
In *First International Colloquium on Theoretical Aspects of Computing*. UNU-IIST. Guiyang, China.


Marcano, R. and Levy, N. (2002).
Using B formal specifications for analysis and verification of UML/OCL models.
In *Workshop on consistency problems in UML-based software development. 5th International Conference on the Unified Modeling Language*, Dresden, Germany.


Petit, D., Poirriez, V., and Mariano, G. (2004).
The B method and the component-based approach.
Journal of Design & Process Science: Transactions of the SDPS, 8(1):65–76. ISSN 1092-0617.


Rocheteau, J., Colin, S., Mariano, G., and Poirriez, V. (2004).
Évaluation de l'extensibilité de PhoX : B/PhoX un assistant de preuves pour B.
In *Journées Francophones pour les Langages Applicatifs*, pages 139–153.

Questions ?

History

The core tools (BCaml)

UML/B

BPhoX

Code generation

Conclusion & perspectives

References

Questions