HATS:
Highly Adaptable & Trustworthy Software Using Formal Models

Reiner Hähnle

Chalmers University of Technology, Gothenburg, Sweden

Sophia-Antipolis, 23 October 2008

Hats Facts

HATS : Highly Adaptable & Trustworthy Software Using Formal Models

Proposal Data

- FP7 FET focused call Forever Yours
- Submitted 8 April 2008
- Integrated Project, academically driven
- 8 academic partners, 2 industrial research, 1 SME
- 7 countries
- Negotiations concluded, project start 1 February or 1 March 2009
- 730 PM, EC contribution 5,27 M€ over 48 months

Consortium & Lead Researchers

Hähnle (Coord.)	Chalmers Tekniska Högskola	SE
Johnsen, Steffen	Universitetet i Oslo	NO
Dam, Gurov	Kungliga Tekniska Högskolan	SE
Puebla, Barthe	Universidad Politécnica de Madrid / IMDEA-Sw	ES
Poetzsch-Heffter	University of Kaiserslautern	DE
Sangiorgi, Zucca	Università di Bologna e Genova	IT
De Boer	Centrum voor Wiskunde en Informatica	NE
Østvold	Norsk Regnesentral	NO
Diakov	Fredhopper	NE
Muthig	Fraunhofer IESE	DE
Clarke, Piessens	Katholieke Universiteit Leuven	BE

Technological and Industrial Context of HATS

Software Dynamics: Adaptability

- Large software systems are extremely long-lived
- Variability: software must work in plethora of deployment scenarios
- Evolvability: frequent and unanticipated changes of requirements

Technological and Industrial Context of HATS

Software Dynamics: Adaptability

- Large software systems are extremely long-lived
- Variability: software must work in plethora of deployment scenarios
- Evolvability: frequent and unanticipated changes of requirements

Software Quality: Trustworthiness

- Non-functional aspects (security, resources) ever more important
- Challenges: product complexity, composability, concurrency
- Ensure and maintain intended behaviour

Technological and Industrial Context of HATS

Software Dynamics: Adaptability

- Large software systems are extremely long-lived
- Variability: software must work in plethora of deployment scenarios
- Evolvability: frequent and unanticipated changes of requirements

Changes rest on behavioural assumptions ↕ New functionality causes security issue

Software Quality: Trustworthiness

- Non-functional aspects (security, resources) ever more important
- Challenges: product complexity, composability, concurrency
- Ensure and maintain intended behaviour

Technological and Industrial Context of HATS

Software Dynamics: Adaptability

- Large software systems are extremely long-lived
- Variability: software must work in plethora of deployment scenarios
- Evolvability: frequent and unanticipated changes of requirements

Changes rest on behavioural assumptions ↕ New functionality causes security issue

Software Quality: Trustworthiness

- Non-functional aspects (security, resources) ever more important
- Challenges: product complexity, composability, concurrency
- Ensure and maintain intended behaviour

Software Economics: Cost-Efficiency

- Automation: far-reaching tool support is essential

High Adaptability + High Trustworthiness: Challenges

Technology Context

- Distributedness
- Concurrency
- Object-Orientation
- Invasive composition

Adaptability

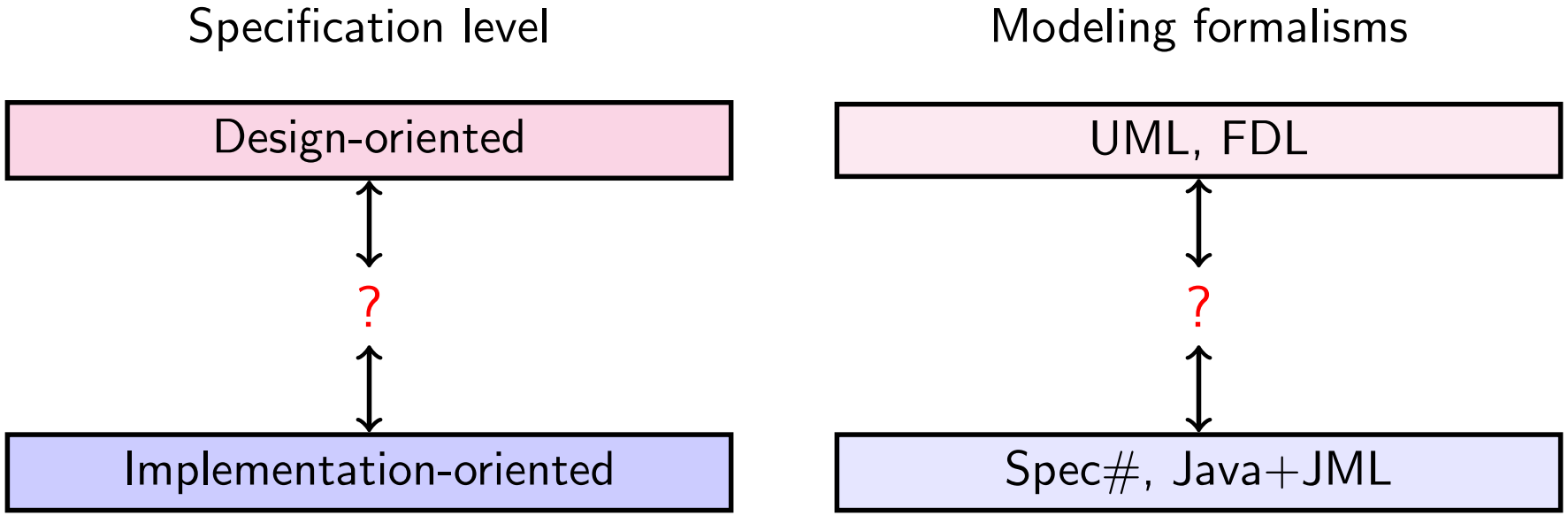
- Many deployment scenarios
- Rapidly changing requirements
- Unanticipated requirements

Trustworthiness

- Correctness
- Security
- Reliability
- Efficiency

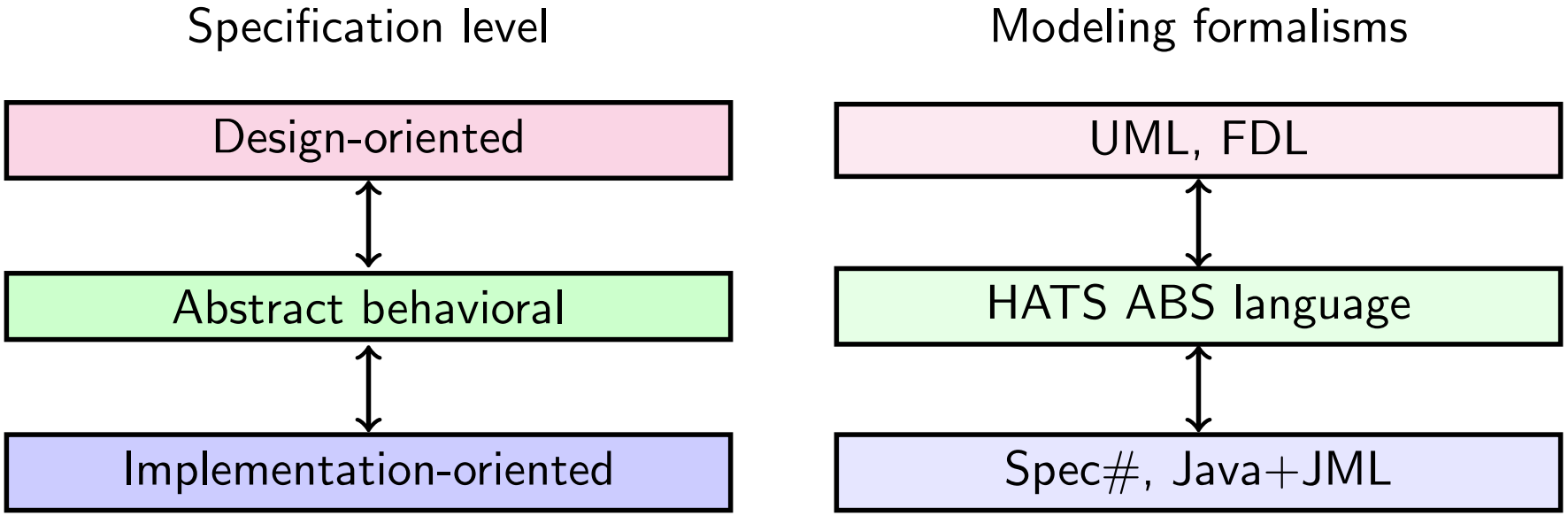
The Modelling Gap

How to precisely model large, distributed systems?



The Modelling Gap

How to precisely model large, distributed systems?



Proposed Solution

A tool-supported formal method for building highly adaptable and trustworthy software

Proposed Solution

A tool-supported formal method for building highly adaptable and trustworthy software

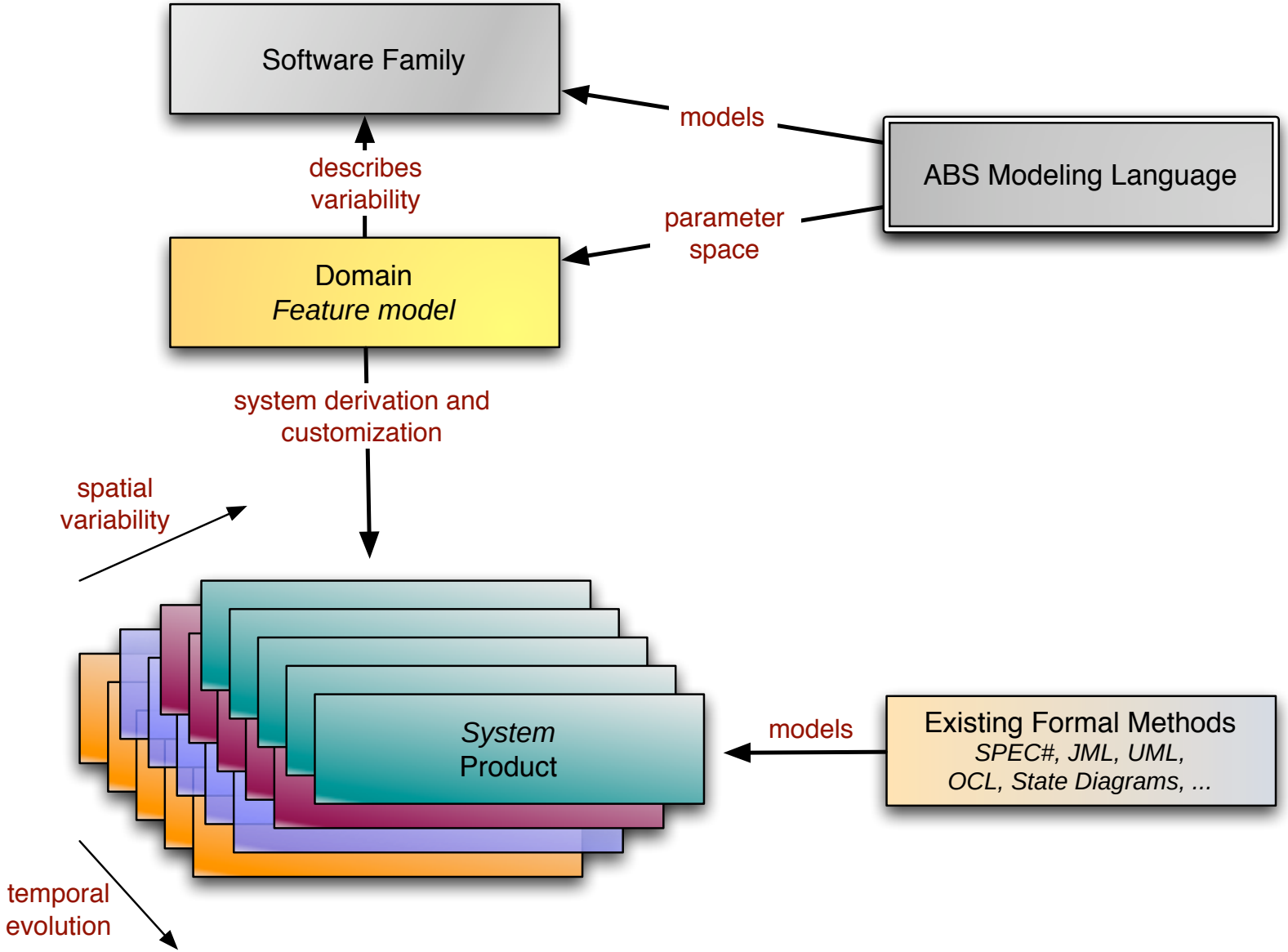
Ingredients

1. Executable modeling language for adaptable software: Abstract Behavioral Specification(ABS) language
2. Integrated framework and tool architecture
3. Tool suite for analysis and development:

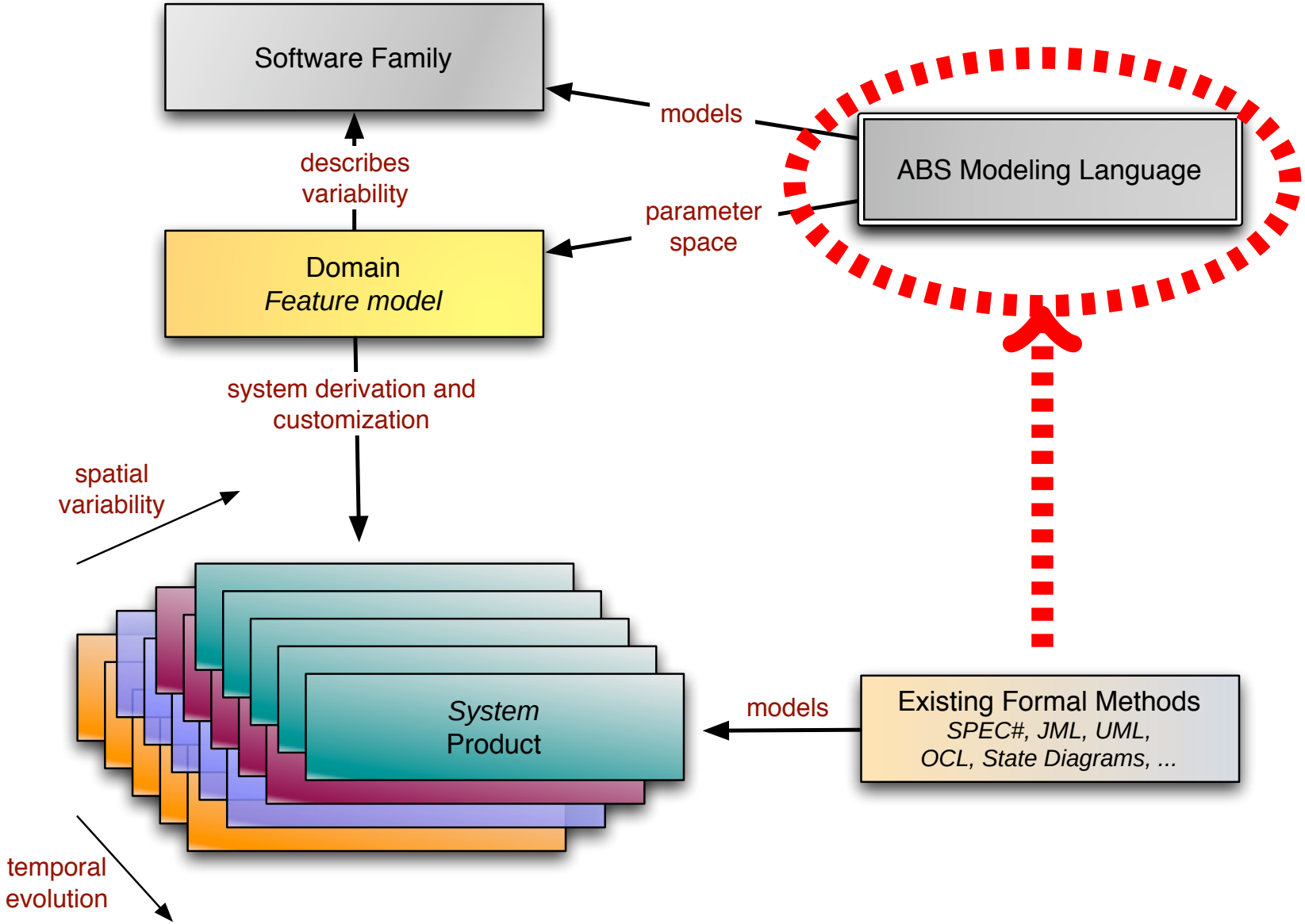
   Hard feature consistency, data integrity, security, correctness, code generation

   Soft visualization, test case generation, specification mining, type checking

Formalising Software Family-Based Development



Scaling Formal Methods to Adaptable Systems



Methodology

Advanced software validation tools need rigouros and unambigous models

Methodology

Advanced software validation tools need rigouros and unambigous models

Abstract Behavioral Specification Language

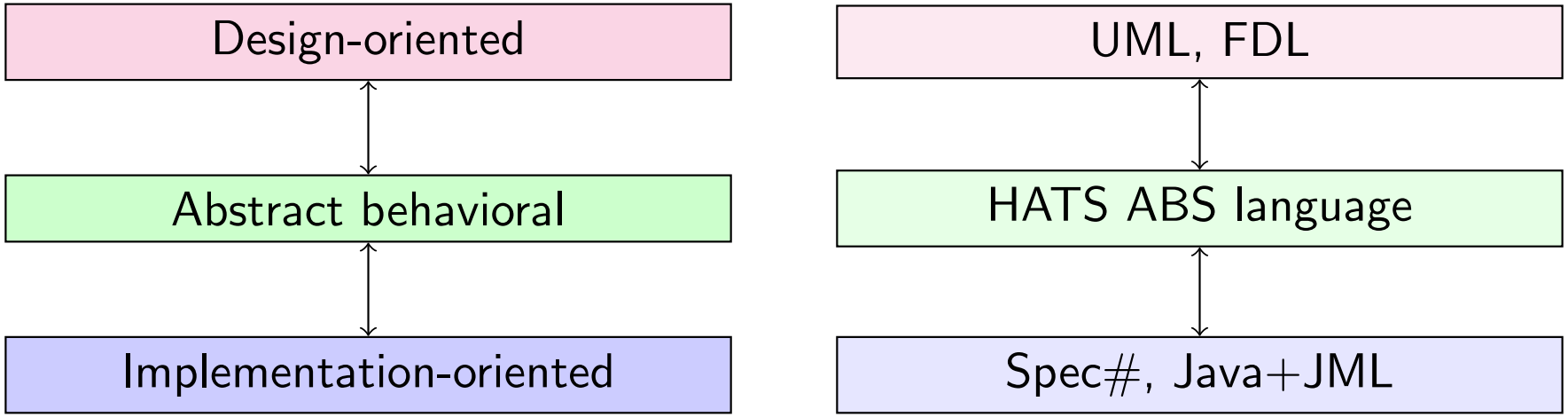
- Adaptability concerns drive its design
- Formalises successful SWPF development method
- Behavioral model: concurrency, composability, modularity, deployment
- Abstract away from programming languages, system architecture

Methodology

Advanced software validation tools need rigouros and unambigous models

Abstract Behavioral Specification Language

- Adaptability concerns drive its design
- Formalises successful SWPF development method
- Behavioral model: concurrency, composability, modularity, deployment
- Abstract away from programming languages, system architecture



Methodology

Advanced software validation tools need rigouros and unambigous models

Abstract Behavioral Specification Language

- Adaptability concerns drive its design
- Formalises successful SWPF development method
- Behavioral model: concurrency, composability, modularity, deployment
- Abstract away from programming languages, system architecture

Validation/verification methods developed in tandem with ABS

- Verification exploits ABS language features
- Native support of variability
- Evolvability: incremental algorithms, code generation, specification mining

System Derivation and Tool Architecture

ABS model and policy

Executable code

Our Approach: ABS models annotated with a policy

1. Does the ABS model comply with the policy?
   - security: confidentiality, application-specific policies
   - resource consumption limitations
   - correctness properties
2. Does the code comply with the ABS model, and the policy?
   - Much stronger guarantees possible!
   - Hard to show for hand-written (legacy) code

Our Proposal: compilation and mining

- Policy-preserving code generators
- Mining ABS specifications from legacy code
- Compile variability of ABS model into code-level verification heuristics

Novel Aspects

Adaptability as the driver and touchstone for a formal method

- First-class support (primitives) in the specification language
- Determines architecture and portfolio of verification methods

Novel Aspects

Adaptability as the driver and touchstone for a formal method

- First-class support (primitives) in the specification language
- Determines architecture and portfolio of verification methods

Leverage successful adaptive approach to formal setting

- Rigorous models of software product families (SWPFs)
- Enable tool-supported validation for SWPF

Novel Aspects

Adaptability as the driver and touchstone for a formal method

- First-class support (primitives) in the specification language
- Determines architecture and portfolio of verification methods

Leverage successful adaptive approach to formal setting

- Rigorous models of software product families (SWPFs)
- Enable tool-supported validation for SWPF

A formal basis for development of "eternal systems"

- Prepared to integrate unforeseen requirements
- Incremental — model-centric — abstract, yet behavioural

Advancing the State of the Art

- Formal basis for SWPF-based development
- Precise behavioural specification of compositional, distributed systems
- Reduced cost of creating and maintaining formal specifications
  - Modeling variability (deployment space)
  - Modularity and compositionality (reusability upon dynamic change)
  - Specification mining, white-box test case generation, visualization
  - Formal models of self$*$ properties
- Verification of compositional and concurrent systems
- Verification of end-to-end properties: security, resources, correctness

Expected Impact

	End-of-project	Long term
Scientific	– Formal methods ready to specify and verify behaviour of adaptive software – Incremental and modular methods	– Formal basis for the convergence of programming languages and control theory – Routine verification of the implementation of large, distributed, adaptable systems
Technological	– Integrated tool architecture for wide range of analysis and verification methods – Formalized SWPF-based development clearly showing the benefits of FM	– Routine usage of formal methods in development of long-lived software – Can ensure far-reaching correctness, security and resource guarantees – Multitude of component libraries specified with ABS

Questions?

The Case for a new Behavioural Specification Language

Essential language features to realize goals

- Formal and integrated semantics
- Layered architecture: simplicity, separation of concerns
- Executability: rapid prototyping, visualization
- Abstraction: underspecification, non-determinism, feature model
- Variation points in the behavioural interfaces of components
- Behavioural types (concurrency/ownership/session)
- Module concept for dealing with invasive composition
- Realistic, yet language-independent concurrency model

Usability, industry-strength tools demand first-class support of features

Technology Basis of ABS

No existing language fulfills these requirements ...

Technology Basis of ABS

No existing language fulfills these requirements ...

... but we build on existing work mainly done by consortium members:

- The concurrency model of CREOL
- Behavioural type systems for OO languages and concurrency
- Logic-based modeling of resources and security
- The CoBox model for modular components
- Verification of open, distributed systems
- The idea of variation points taken from SWPFs
- The Feature Description Language (FDL)