Formal validation of TASTE designs with the BIP framework

Iulia Dragomir

1Universite Grenoble Alpes – VERIMAG, France

ESROCOS workshop

June 21, 2017
Shortcomings of system design in practice

- Requirements in natural language, therefore ambiguous or incomplete

- Large, reactive, component-based designs in graphical semi-formal languages
  → ambiguous or unspecified operational semantics
  → erroneous implementations due to different possible interpretations

- A posteriori validation of the implementation
  → testing is incomplete (e.g., autonomous systems operating in unknown environment)

- Increased development effort, costs, resources
ESROCOS project: TASTE language and tool

- Targets the model-based development of heterogeneous, reactive, discrete embedded systems
- The language is based on several modeling formalisms (ASN.1, AADL, SDL, Simulink, etc.) or programming languages (C, Ada)
- A TASTE design consists of:
  - Data view
  - Hierarchical interface views (software architecture and behavior)
  - Deployment view
  - Concurrency view computed from the above
- The tool-chain allows for:
  - Well-formedness and typing analysis of a design wrt the used modeling formalisms
  - Real-time scheduling analysis (e.g., Cheddar)
  - Code generation and deployment (in C/C++, Ada) wrt an execution platform
  - Simulation, debugging and testing
Rigorous system design

- **High-level formal modeling frameworks** encompassing heterogeneous computation models
  → **model- and component-based**

- **Scalable validation and verification** methods and tools

- **Performance analysis** and **resource optimization**

- **Correct-by-construction** transformations towards implementations (code)

The BIP framework offers support for rigorous system design.
Outline

1. Context and motivation
2. The BIP framework
3. Formal validation of TASTE design with BIP
4. Conclusion
Outline

1. Context and motivation
2. The BIP framework
3. Formal validation of TASTE design with BIP
4. Conclusion
The BIP framework

- **Component-based design:**
  - **Behavior:** atomic functional units (automata, code, etc.)
  - **Interactions:** cooperation and coordination between actions of behavior
  - **Priorities:** scheduling between interactions

- **Model-based design:**
  - **Heterogeneity:** execution, interaction, abstraction, etc.
  - **Minimal set of constructs and principles**
  - **Automated validation, verification and performance analysis**
  - **Automated code generation for given platforms**
The BIP tools

- Language factory (C, DOL, etc.)
- Simulator (real-time, etc.)
- Code generation
- SMC-BIP
- iFinder
Outline

1. Context and motivation
2. The BIP framework
3. Formal validation of TASTE design with BIP
4. Conclusion
The formal validation workflow with BIP

- Assumptions
- Safety requirements
- iFinder
- Nominal BIP model
- Faults BIP model
- Simpler (BIP Compiler & Engines)
- SMC-BIP
- PB-LTL/MTL requirements

- OK/KO verdict
- Invariant(s)
- OK/KO verdict, counterexample
- Probability estimation
- Statistical parameters

Iulia Dragomir (VERIMAG)
Formal validation of TASTE designs with the BIP framework
June 21, 2017
TASTE2BIP model translator

- New tool generating BIP models from full TASTE designs
- Tool input: data / interface / deployment / concurrency views, SDL state machines
- Tool output: nominal BIP model, possibly including FDIR components

Principles of translation:
- Data view, interface view and SDL state machines give a timed automata network
- Deployment and concurrency views refine the timed automata network into a HW/SW BIP model
- Enforcement of timing constraints, e.g., period, wcet
iFinder verification tool

- Check safety properties of a system by separate analysis of components and their composition – compositional invariant-based method
  - component invariants
  - interaction invariants
  - history clocks constraints

- Current black-box tool to be redesigned

- New tool features:
  - User-given invariants for property verification (assumptions)
  - Modularity of invariant computation and generation
  - Interaction with different SAT solvers (e.g., Yices)

![Diagram of the verification process](image-url)
BIP compiler and engines

- (C++) Code generation and simulation engines
- Extensions:
  - Timing constraints: real-time executions
  - Stochastic transitions: probability to fire a transition depending on a user-given probability law $\lambda$
  - Faults model: identification of faults, timed fault, probability of faults to occur, etc.
SMC-BIP analysis module

- **Statistical model-checking** of a stochastic system $S$ and a requirement $\varphi$:
  - **Qualitative question**: $S \models P \geq \theta \varphi$
    $\rightarrow$ yes or no answer with a confidence bound
  - **Quantitative question**: $P(S \models \varphi)$
    $\rightarrow$ the estimation can be bounded, susceptible to mistakes

- **Extensions**:
  - Real-time, stochastic behavior
  - Richer property logic for formalizing requirements, e.g., MTL
Outline

1. Context and motivation
2. The BIP framework
3. Formal validation of TASTE design with BIP
4. Conclusion
Contribution to ESROCOS

- **BIP**: a framework formalizing robotic systems designed in TASTE
- **TASTE2BIP**: an automated tool to generate BIP models from TASTE
- A set of tools to verify and validate TASTE designs via BIP
  - BIP compiler for C++ code generation
  - BIP engines for simulation
  - iFinder for safety requirements verification
  - SMC-BIP for statistical model-checking of performance requirements

![Diagram showing the flow from TASTE design to BIP models and verification tools]

Iulia Dragomir (VERIMAG)
Formal validation of TASTE designs with the BIP framework
June 21, 2017
Ongoing work

- Invariant generation
- Safety analysis
- Assumption and requirement parsing
- Counterexample analysis

TASTE design

- Definition of the operational semantics and translation principles
- Proof-of-concept on example
- Implementation

Nominal BIP model

- Invariant generation
- Safety analysis
- Assumption and requirement parsing
- Counterexample analysis

TASTE2BIP

- Nominal BIP model
- Faults BIP model

Fault model extension

iFinder

- OK/KO verdict, counterexample

Invariant(s)

- Real-time stochastic monitoring
- MTL requirements

Simulator (BIP Compiler & Engines)

OK/KO verdict, counterexample

SMC-BIP

BY

PB-LTL/MTL requirements

OK/KO verdict

Probability estimation

Statistical parameters

- Real-time stochastic monitoring
- MTL requirements

- Fault model extension

Thank you!
Questions?

Iulia Dragomir (VERIMAG)
Ongoing work

- Invariant generation
- Safety analysis
- Assumption and requirement parsing
- Counterexample analysis

- Definition of the operational semantics and translation principles
- Proof-of-concept on example
- Implementation

- Nominal BIP model
- Faults BIP model

- OK/KO verdict, counterexample
- Invariant(s)

- Simulator (BIP Compiler & Engines)
- Statistical parameters
- Real-time stochastic monitoring
- MTL requirements

- PB-LTL/MTL requirements
- OK/KO verdict
- Probability estimation

Thank you!
Questions?

Iulia Dragomir (VERIMAG)