Formal Development for Railway Signaling Using Commercial Tools

Alessio Ferrari, Alessandro Fantechi
Stefano Bacherini and Niccolò Zingoni

General Electric Transportation Systems, Firenze, Italy
Università di Firenze (DSI), Firenze, Italy

Background

The rapid and wide-spread diffusion of model based development practices in the safety-critical industry have seen the clear establishment of the Simulink/Stateflow platform as a de-facto standard for modeling and code generation. This success is mainly due to the several capabilities of the tool-suite and to the effective and engineering-friendly modeling languages. The large number of built-in blocks provided by Simulink, together with the notable capabilities of Stateflow statecharts, allows fast development of prototype applications which can be simulated and directly analyzed with the support of the Matlab environment.
The General Electric Transportation Systems (GETS) railway signaling division of Florence, inside a long-term effort of introducing formal methods to enforce product safety, decided to adopt the Simulink/Stateflow tool-suite to exploit model based development and code generation within its own development process.

Certification Problem

Code generators provided for the tool-suite are not certified for railway software development, this complicating their adoption in this domain. In this case a question has to be raised concerning the reliability of the auto-coding tools: how to ensure that the behavior of the generated code is consistent with the corresponding model behavior?

Formal Verification Problem

The languages used by Simulink and Stateflow are not formally specified and their semantics is essentially given by the simulation engine itself. This lack of formal semantics increases the difficulty of defining an effective formal verification strategy: how to formally verify a model whose language is not formally defined?

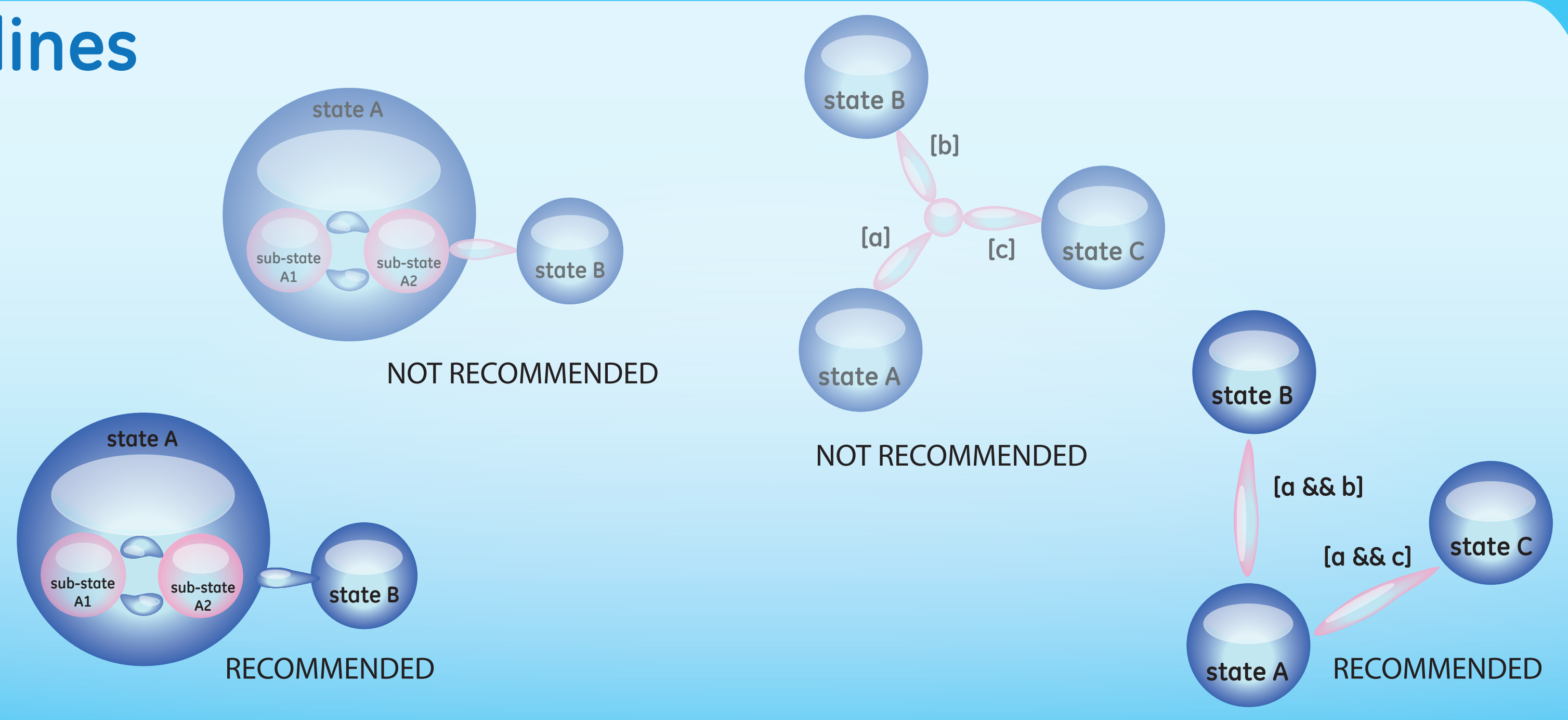
Certification Solution

Formal Verification Solution

Modeling Guidelines

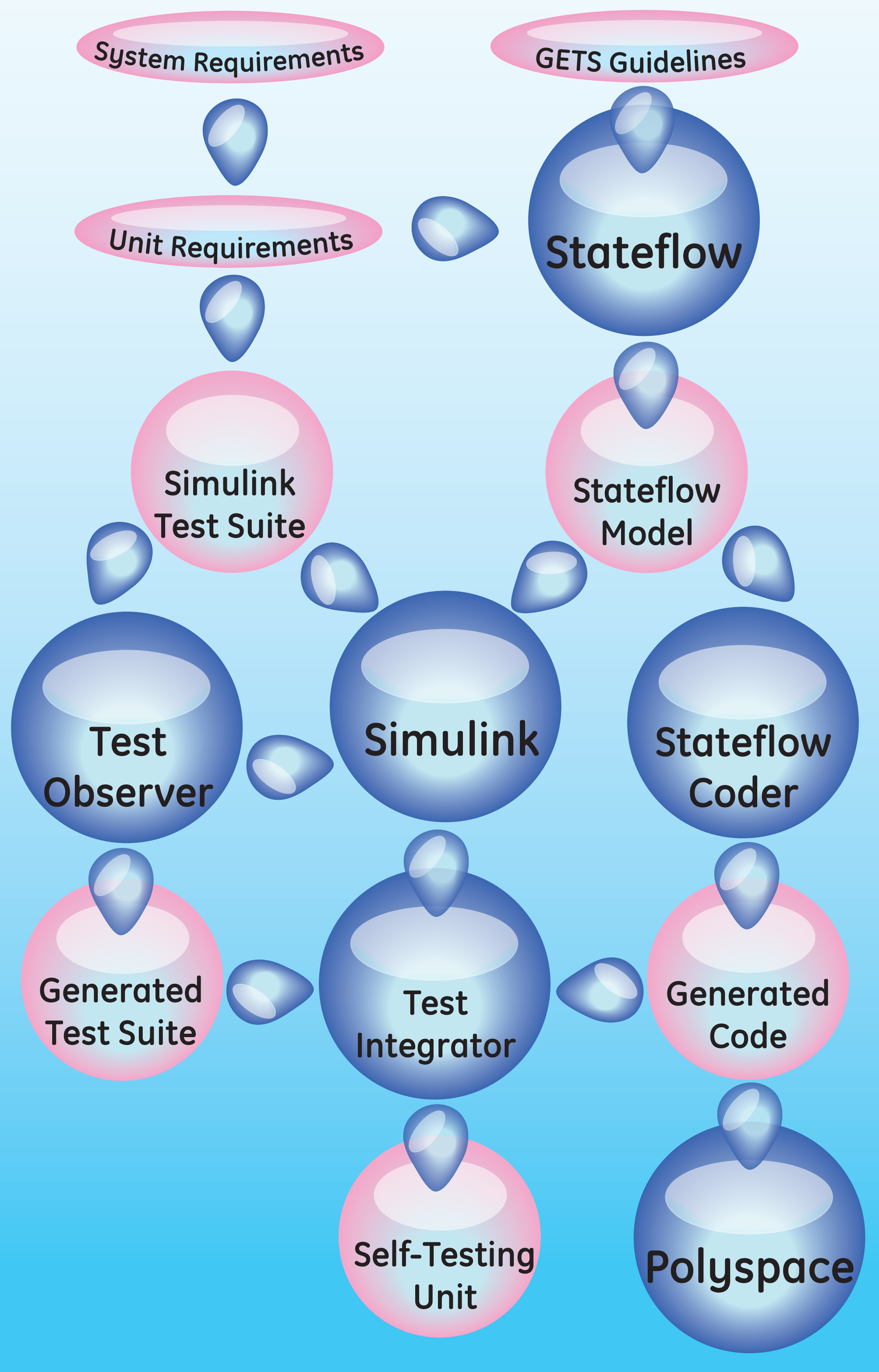
A set of modeling guidelines have been introduced by GETS in order to restrain the semantics of the tools and guarantee generation of readable, structured and traceable code. The semantics restrictions are also deemed to allow effective model analysis and formal verification.

Given a set of system-level functional requirements, these can be partitioned into separate sets of unit requirements and then formalized into Stateflow models according to the GETS guidelines. Each model represents an independently verifiable system component.



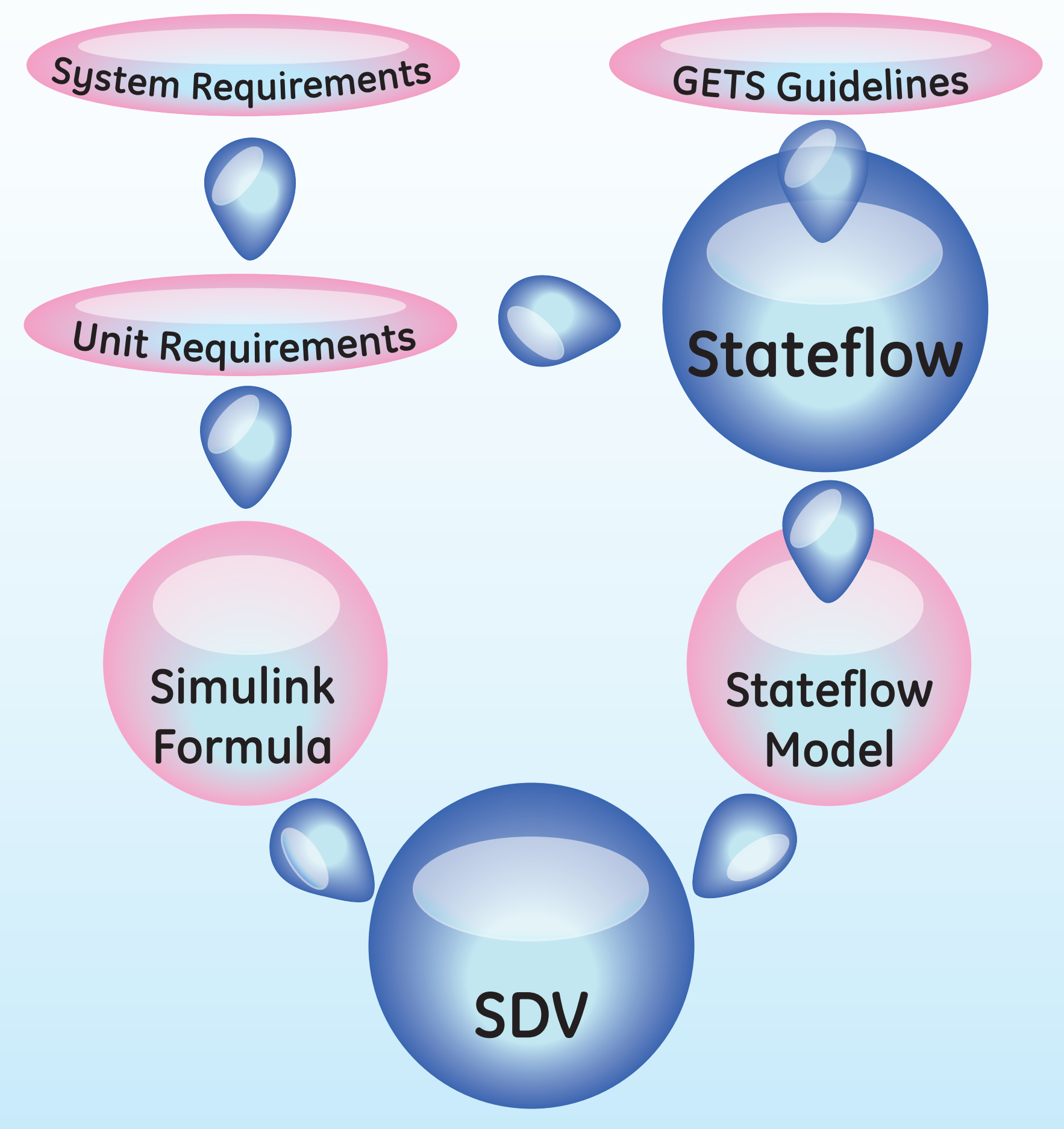
Model Based Testing and Abstract Interpretation

Our solution for the certification problem consists in two steps, namely model based testing and abstract interpretation.
Unit testing based on requirements coverage is performed on the models through the Simulink environment, and during test execution a test observer is used to register the test-suite input data and the test results. The registered test-suite is executed on the auto-coded unit and results are automatically compared.
Finally, the unit is analyzed through the Polyspace tool, based on abstract interpretation, in order to increase the confidence on the correctness (in particular, absence of runtime errors) of the generated code.



Property Proving

Formal verification is provided at unit level, using a tool developed by the Mathworks called Simulink Design Verifier (SDV), a property proving engine based on Prover technology, currently in its preliminary phase for adoption within industries.
Model units are defined in the form of Stateflow statecharts and unit specifications are expressed using the Simulink formalism, that allows to specify assertions using block diagrams. The tool allows the verification of the Stateflow model against the Simulink formula.



Case Study

The approach has been experimented during the development of the software for an Automatic Train Protection system called SSC/SCMT Baseline 3. The original natural language specification have been formally represented through a Stateflow specification of 21 Stateflow charts.

The generated code consisted of 150K LOC, and both the model based testing (327 test scenario covering 100% of the functional requirements) and abstract interpretation steps have been performed during the verification process. Though the effort of manually defining the test scenarios for the models can be compared with the one of defining tests for hand-crafted code, the abstract interpretation step, basically automatic, ensures a confidence on the absence of runtime errors that can not be achieved with traditional testing techniques.

Property proving is still at its experimental phase and it has been provided for three core charts of the system. The major effort in this case was the proper definition of the Simulink formulas, that took about the same time required for defining the Stateflow charts to be verified. Nevertheless these formulas normally have a reusable structure, and we are currently investigating which are the underlying patterns that can be used for their definition.