Egon Börger (Pisa)

The ASM Method Integrates Validation and Verification

at Different Abstraction Levels along the ASM Refinements

Talk at IVOIRE Workshop @ iFM 2022 (Lugano 7.5.2022)

boerger@di.unipi.it

Three Questions from Michael Leuschel

Michael asked me about my experience with:

- the limits of formalization,
- the difference between validation and verification,
- the range of what can be formally checked and what must remain informal

In their Validation Obligations paper Mashkoor/Leuschel/Egyed state:

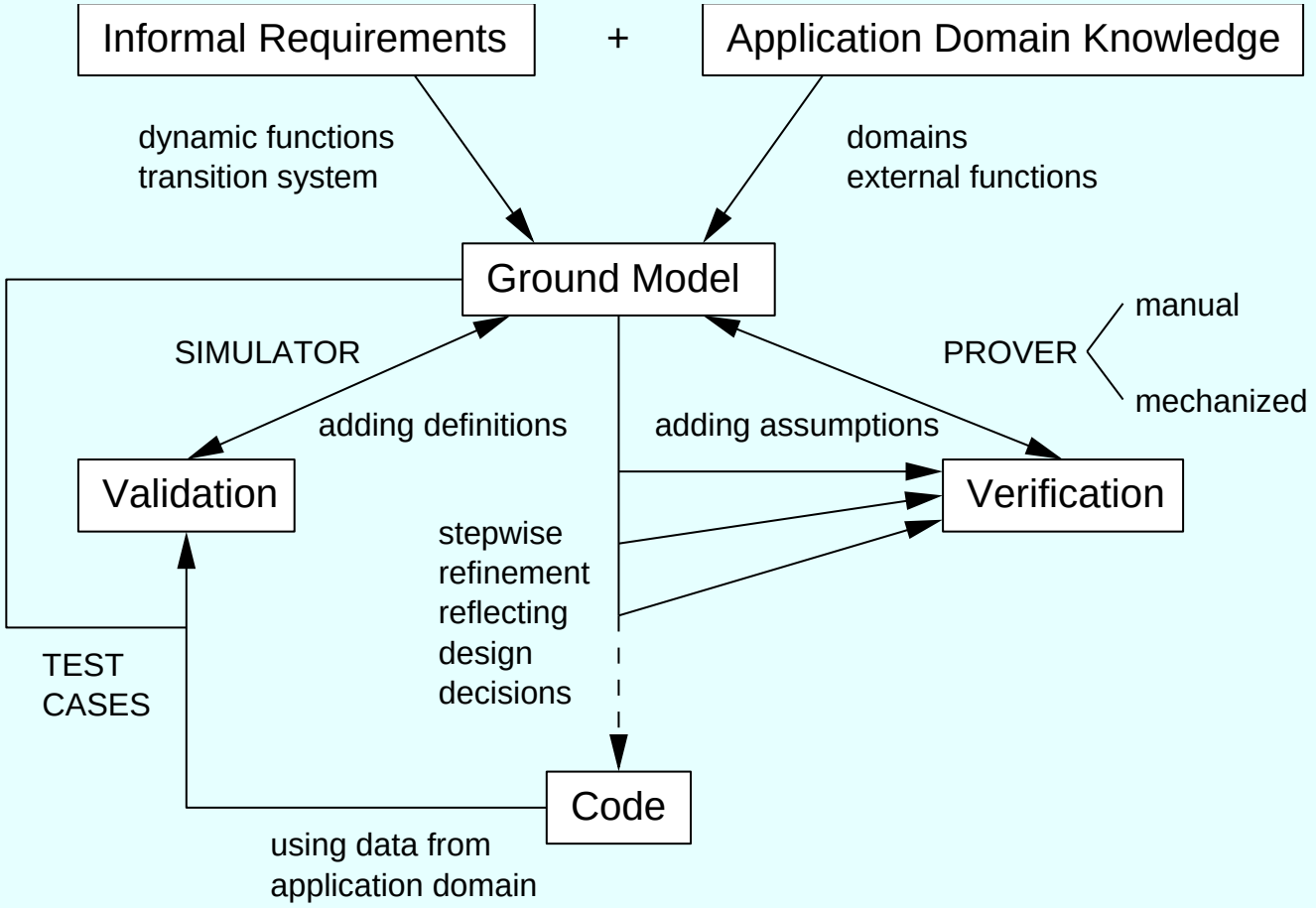
> ... validation is usually postponed until the latest stages of the development, when models can be automatically executed. Thus mistakes in requirements or in their interpretation are caught too late: usually at the end of the development process.

My first answer is that this does not hold for the ASM method (Abstract State Machines).

The following slides give a more detailed answer to Michael's questions.

The ASM Method integrates Validation and Verification

... at each level of abstraction along ASM refinement chain (since 1989)



Note the iterative (not necessarily linear) character of the ASM method.

Various Reasons for Validation of ASMs

The *executable character of ASMs* from the very beginning lead us to use validation of models at given levels of abstraction, for example:

- Validation of Ground Models (also called *primary models*): overcoming verification limits via *inspection* & conceptual or physical validation
  - defining standards for ISO, IEEE, ITU-T, ECMA, OASIS, OMG
  - validation of scenario-based system development (Siemens 1998-1999) using AsmWorkbench (G. DelCastillo)
- Validation at different levels of abstraction
  - to experiment with design ideas at high-level of abstraction
    - Prolog interpreter ASM at Quintus (D.Bowen 1990)
    - CoreASM reference implementation of an S-BPM engine (2021)
    - check examples before embarking on a correctness proof (2013)
  - comparison of runs of models at different abstraction levels
    - Java/JVM AsmGofer (J.Schmid) and Sun's machines (1999-2001)
    - AsmL models and code: runtime exec monitored by spec (MS 2000)

Ground Model: Validation to overcome Formalization Limits

- Ground model: a precise, correct and complete description of the requirements (including domain assumptions), i.e. of the system behaviour as seen at the level of abstraction of the application domain
- Epistemological limits of formalization: a ground model cannot be proved (in the mathematical sense) to be correct and complete since
  - it explains how the intended system behaviour "relates to the affairs of the world it helps to handle" (Naur 1985)
  - the 'affairs of the world' have no formal character (no infinite regress)
- Inspection to *check adequate relation bw real-world and ground model* can provide confidence in the model's correctness and completeness
  - if the model is formulated in terms both parties understand—domain experts and software engineers— to follow the (conceptual and/or mechanical) validation of model features during the inspection
  - NB importance of DIRECT (Aristotle: *evident*) relation between linguistic model terms and the world features they represent

Scenario-Based ASM Ground Model Validation (Siemens)

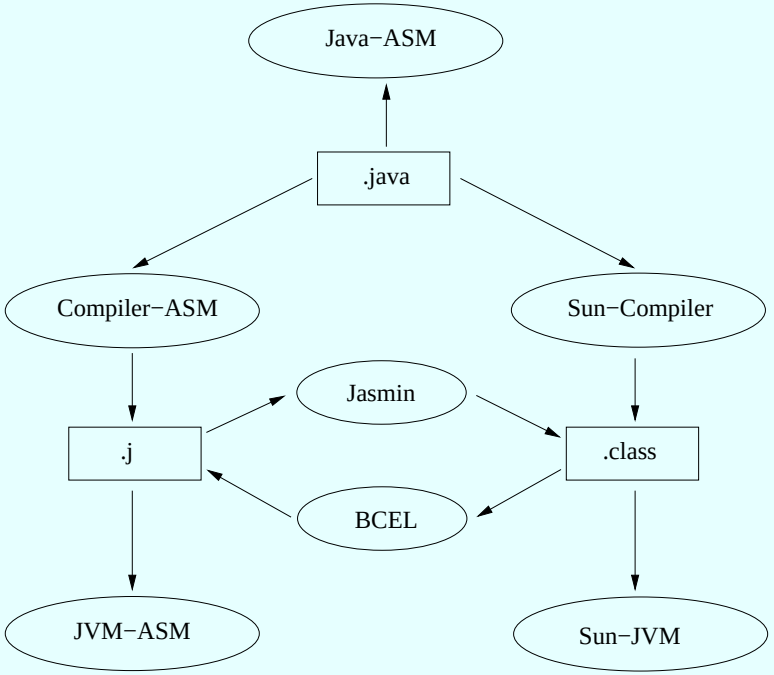
- requirements for a railway timetable construction and validation program were given by a set of concrete schedules and data of train runs in two major German local transportation areas
- goal: design and implement railway process model component for reqs
- *various ASMs were built and validated*: inspecting the models and running all scenarios on ASM Workbench (G. Del Castillo)
- *final ASM refined by Asm2C^{++} compiler* (J. Schmid)
- documentation of ASMs allowed model reuse by ASM refinement when a change request appeared:
  - *we integrated the additional requirement by a model refinement*, revalidated and recompiled the new model so that the new code was again in sync with the refined requirements model

Code worked for years, without failure and without further maintenance, in the Vienna transportation system (until a completely new system was installed). NB there was nothing to verify!

ASM Validation of High-Level Design Ideas: some examples

- D. Bowen programmed (in Prolog) Börger's Prolog-Core ASM for user-defined predicates and used it to run experiments with new Prolog features in the given Quintus-Prolog environment (1990)
- Fleischmann et al. use a CoreASM model of an (extension of Metasonic's) S-BPM engine as reference implementation to experiment with additional features and to implement an industrial-strength workflow engine in $C^{\#}$ (2022)
- Gervasi and Riccobene, for a formalization test in a Dagstuhl seminar (2013), used CoreAsm and ASMeta to (in)validate a famous algorithm: a couple of runs detected some hidden assumptions—nota bene in the Termination Detection Protocol proposed by Dijkstra "to demonstrate how the algorithm can be derived".
  - advice: test (and thereby illustrate) your definitions and statements by examples before embarking on a proof

Mixing/Comparing Runs of AsmGofer/Sun Machines



- Java-ASM reads and executes Java source code (suffix .java)
- Compiler-ASM compiles from .java to .j (Jasmin bytecode, textual)
- SunCompiler generates .class files (binary class file representation)
- JVM-ASM resp.Sun-JVM read input in Jasmin syntax resp. class files

Test runs revealed various bugs (communicated to & repaired by Sun).

Runtime Execution Monitored by AsmL-Spec (MS 2000)

AsmL-specs compiled and interoperated with MS runtime env/lgs:

- Given an AsmL-machine M and code C (an implementation), test externally visible C-behaviour (implementation) against behaviour of M
  - monitor sequence of external component interactions (passed arguments and returned values)
    - no access to component internals
    - no need to instrument the tested components
  - compare observed behaviour with specified behaviour
    - in case of inconsistency report error (in spec or in code)
- Exploration in a given environment of
  - user-scenarios
  - some desired functionality

Sad note: AsmL has been abandoned.

Good note: explorer idea remains valid (has been used by Windows team)

Conclusion

- Up to today, each time validation of ASM models was in the focus, for the mechanical execution of appropriate refinements of the involved (conceptually executable) ASMs
  - either some available machinery could be used
    - AsmL, ASMeta, CoreASM, ...
  - or some tool support has been developed for the specific purpose, exploiting characteristics (and tool support) of the environment where the models are supposed to run
    - AsmWorkbench, AsmGofer, ...

  so that model validation could use physical model runs.
- ASMeta since 2008 has a dedicated tool to validate AsmetaL specifications by scenarios.

References

- R. F. Stärk and J. Schmid and E. Börger: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer 2001.
- E. Börger and P. Päppinghaus and J. Schmid: Report on a Practical Application of ASMs in Software Design. LNCS 1912 (2000) 361–366.
- M.Elstermann et al:The Combined Use of the Web Ontology Language (OWL) and Abstract State Machines(ASM) for the Definition of a Specification Language for Business Processes. LNCS 12750 (2021) 283-300.
- V. Gervasi and E. Riccobene: From English to ASM: On the process of deriving a formal specification from a natural language one. Dagstuhl Report 3 (9) 2014, 85–90.
- E. Börger and A. Raschke: Modeling Companion for Software Practitioners. Springer 2018
  http://modelingbook.informatik.uni-ulm.de
- E. Börger and R. Stärk: Abstract State Machines. Springer 2003.

References for Tool Support

⋮

- Workbench https://dl.acm.org/profile/81100469826
  http://web.eecs.umich.edu/gasm/papers/workbench.html
- AsmGofer https://tydo.eu/AsmGofer/
- CoreASM https://github.com/coreasm/
- Asmeta https://asmeta.github.io/
- CASM https://casm-lang.org/

Copyright Notice